A FRAMEWORK FOR ANALYZING AND MITIGATING THE VULNERABILITIES OF COMPLEX SYSTEMS VIA ATTACK AND PROTECTION TREES

DISSERTATION

Kenneth S. Edge, Major, USAF

AFIT/DS/ENG/07-13

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/DS/ENG/07-13

A FRAMEWORK FOR ANALYZING AND MITIGATING THE VULNERABILITIES OF COMPLEX SYSTEMS VIA ATTACK AND PROTECTION TREES

DISSERTATION

Presented to the Faculty

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

Kenneth S. Edge, BSEE, MSEE

Major, USAF

July 2007

AFIT/DS/ENG/07-13

A FRAMEWORK FOR ANALYZING AND MITIGATING THE VULNERABILITIES OF COMPLEX SYSTEMS VIA ATTACK AND PROTECTION TREES

Kenneth S. Edge, BSEE, MSEE
Major, USAF

Approved:

Richard A. Raines, PhD Committee Chairman	11 Jul 07 Date
William P. Baker, PhD Dean's Representative	20 Jul 07 Date
Rusty O. Baldwin, PhD Committee Member	16 Jul 07 Date
Michael R. Grimaila, PhD Committee Member	16 JUL 07 Date
Robert W. Bennington, PhD Committee Member	17 Jul 07 Date
Christopher E. Reuter, PhD Committee Member	17 Jul 07 Date

Accepted:

MUThomas Marlin U. Thomas, PhD Dean	1 Aug 07 Date

AFIT/DS/ENG/07-13

A FRAMEWORK FOR ANALYZING AND MITIGATING THE VULNERABILITIES OF COMPLEX SYSTEMS VIA ATTACK AND PROTECTION TREES

by

Kenneth S. Edge, BSEE, MSEE
Major, USAF

Richard A. Raines, Advisor

In recent years, *attack trees* have been developed to describe processes by which malicious users attempt to exploit or break complex systems. Attack trees offer a method of decomposing, visualizing, and determining the cost or likelihood of attacks. Attack trees by themselves do not provide enough decision support to system defenders. The defenders need methods to determine which protections to implement and where to place them in the system to mitigate the vulnerabilities found. This research develops the concept of using *protection trees* to offer a detailed risk analysis for the protection of a system. In addition to developing protection trees, this research improves the existing concept of attack trees as well as developing rule sets for the manipulation of metrics used in the security of complex systems.

This research specifically develops the framework for using an attack and protection tree methodology to analyze the security of complex systems. To accomplish this, the structure of attack trees is extended and modified to create the concept of protection trees. To validate the effectiveness of the methodology, the Schematic Protection Model (SPM) is used. The SPM is extended and applied to verify that a

system protected using the attack and protection tree methodology is safe. To demonstrate the general usefulness of this novel methodology, it is used to analyze the security of several varied domains including computer networks, online banking, homeland security, and mobile ad hoc networks.

Acknowledgments

First, I would like to thank God for blessing me with all He has given to me in life. Although I don't always understand why He has put me on certain paths in my journey through life, I know it is all part of His plan for me. I also want to thank my beautiful and patient wife for helping me through this journey with all she has done for me. My job was easy because of everything she did for our family at home. I also want to thank my children for being accepting of my frequent absences from home during the pursuit of this degree.

I thank my advisor, Dr. Richard Raines, for allowing me the latitude and support to explore different subject areas to finally figure out what I wanted to do. I also would like to thank my committee members, Dr. Baldwin, Dr. Grimaila, Dr. Bennington, and Dr. Reuter, for helping me throughout the research with feedback and ideas. Finally, I would like to thank my fellow students for the experience and insight you have given me into many different facets of the Air Force. The amazing things that the Air Force does are truly because of your hard work.

Table of Contents

Acknowledgments vi
Table of Contents vii
List of Figures xii
List of Tables xv
List of Tables xv
List of Symbols xvi
List of Symbols xvi
List of Abbreviations xix
I. Introduction 1
1.1 Research Motivation 1
1.1.1 Goals. 1
1.2 Research Contribution 2
1.2.1 Framework of Methodology Established 2
1.2.2 Extension of Schematic Protection Model (SPM). 2
1.2.3 Abstraction of Methodology. 2
1.3 Research Assumptions 3
1.3.1 Metrics. 3
1.4 Dissertation Organization 3
II. Background 5
2.1 Attack Trees 5
2.1.1 Threat Logic Trees. 5
2.1.2 Attack Tree Term Introduced. 7
2.1.3 Commercial Implementation of Attack Trees 11
2.1.4 Attack Specification Language. 13

2.1.5 Attack Patterns and Profiles 13
2.1.6 Additional Logic Gates in Attack Trees. 16
2.1.7 Transformation and Equivalence of Attack Trees. 16
2.1.8 Stratified Node Topology. 18
2.1.9 Goal-Inducing Attack Chains. 19
2.1.10 Libraries and Common Knowledge of Vulnerabilities. 20
2.1.11 Automatic Generation of Attack Trees 21
2.2 Attack Metrics 21
2.2.1 Metric Guidelines 21
2.2.2 Attack Classifications. 23
2.2.3 Vulnerability Index. 24
2.2.4 Alternatives to Metrics 25
2.2.5 Empirical Metrics 25
2.3 Protection Trees 27
2.4 Schematic Protection Model (SPM) 28
2.4.1 Safety Question. 28
2.4.2 Schematic Protection Model. 28
2.5 Background in Application Areas 32
2.5.1 Virus Protections 32
2.5.2 Artificial Immune Systems in Computer Virus Detection 44
2.5.3 Mobile Ad Hoc Networks (MANETS) Background 46
2.6 Background Summary 53
III. Framework Development 54

3.1 Attack and Protection Tree Methodology 54

3.1.1 Attack Trees. 54

3.1.2 Protection Trees 62

3.1.3 Generic Attack and Protection Tree Security Analysis. 66

3.2 Summary of Framework Development 70

IV. Applications 72

4.1 Network Security Application 73

4.1.1 Example Protection Tree for a Network Security Application 74

4.1.2 Analysis.. 78

4.1.3 Network Security Application Summary. 79

4.2 Homeland Security 80

4.2.1 Homeland Security Network Application.. 80

4.2.2 Results and Analysis for Homeland Security Application 82

4.2.3 Summary of Homeland Security Application. 88

4.3 Online Banking..88

4.3.1 Online Banking Application. 89

4.4 Mobile Ad Hoc Networks 98

4.4.1 Motivation MANET Application. 99

4.4.2 MANET Application. 101

4.4.3 Results and Analysis. 106

4.4.4 MANET Application Summary. 108

4.5 Research on Specific Vulnerability Detectors and Protections 108

4.5.1 Virus Protections as Legitimate Software Protections.............................. 109

4.5.2 Metamorphism as a Software Protection 114

4.5.3 AIS IDS for MANETS..... 130

4.5.4 A Retrovirus Inspired Algorithm for Virus Detection (REALGO) 138

4.6 Summary of Applications..... 158

V. Validation of Methodology Using the Schematic Protection Model..... 160

5.1 An Introduction to the Schematic Protection Model..... 160

5.2 Implementing the SPM on the Online Banking Application..... 161

5.2.1 Mapping Online Banking Application to SPM..... 161

5.2.2 Initial State of System..... 161

5.2.3 Links between Entities..... 163

5.2.4 Filters 163

5.2.5 History..... 163

5.3 Extension of Authentication in SPM..... 166

5.4 Implementing Filters 169

5.4.1 Implement Challenge and Response System with User Accounts. 169

5.4.2 Harden Bank Server to Prevent Outside Attack. 169

5.4.3 Audit Inside Access to Network. 170

5.4.4 Implement Challenge and Response System in Transfer Network. 170

5.4.5 History with Filters Enabled. 170

5.5 Analysis 174

5.6 Summary..... 175

VI. Conclusion 176

6.1 Research Contributions 176

6.1.2 Publications 178

6.2 Recommendations for Future Research..178

Vita 194

List of Figures

Figure		Page
Figure 2.1.	Threat Logic Tree Example for UNIX System	6
Figure 2.2.	Example Attack Tree to Break into a Safe	9
Figure 2.3.	Text Representation of Attack Tree for an E-mail Security Program	10
Figure 2.4.	Sample Attack Tree Generated by SecurITree	12
Figure 2.5.	Web Server Attack Tree	14
Figure 2.6.	Attack Specification Language BNF	14
Figure 2.7.	Attack Tree Refinement Process	15
Figure 2.8.	Fault Tree Gates	16
Figure 2.9.	Equivalent Attack Trees	17
Figure 2.10.	Associativity of Conjunction Rule	18
Figure 2.11.	Distributivity of Conjunction over Disjunction Rule	18
Figure 2.12.	Stratified Node Topology	19
Figure 2.13.	Attack Tree Demonstrating Computation of Probabilities	21
Figure 2.14.	Fault Tree Generator	22
Figure 2.15.	An Example of a Defense Tree	27
Figure 2.16.	Reverse Transcription in the Human Body	46
Figure 2.17.	Example MANET	47
Figure 2.18.	Host Based Agents in Mobile IDS	49
Figure 3.1.	Example Attack Tree	55
Figure 3.2.	Example Tree Showing Propagation of Metrics in Attack Tree	61

Figure 3.3. Partial Attack Tree to Open a Physical Safe 63
Figure 3.4. A Partial Protection Tree for the Safe Attack 63
Figure 3.5. Generic Attack Tree 67
Figure 3.6. First Iteration of Generic Protection Tree 68
Figure 3.7. Second Iteration Of Generic Protection Tree Optimized for TCF 69
Figure 3.8. Total Cost Factor of Competing Protections 70
Figure 4.1. Partial Network Attack Tree 75
Figure 4.2. Partial Network Protection Tree 76
Figure 4.3. Disable JRIES Attack Tree (High Level) 81
Figure 4.4. JRIES Protection Tree (High Level) 82
Figure 4.5. DDoS Attack Tree 83
Figure 4.6. DDoS Protection Tree 85
Figure 4.7. Pruned DDoS Protection Tree 86
Figure 4.8. Pruned DDoS Attack Tree 87
Figure 4.9. Online Banking Attack Tree 91
Figure 4.10. Initial Online Banking Protection Tree 93
Figure 4.11. Online Banking Protection Tree after First Iteration 94
Figure 4.12. Online Banking Protection Tree Using $200K in Protection Resources 96
Figure 4.13. Attack Tree after Limited Protection 97
Figure 4.14. Pareto Front of Security Solutions 100
Figure 4.15. MANET Attack Tree 102
Figure 4.16. MANET Protection Tree (First Iteration) 105
Figure 4.17. Sensitivity Analysis of Weighting Factor 105

Figure 4.18. Expanded Views of Critical Points in Sensitivity Analysis. 105

Figure 4.19. Second Iteration MANET Protection Tree 107

Figure 4.20. Anatomy of a Simple Encrypted Virus Program 117

Figure 4.21. Two Bytes of Data (0x81 and 0x80) Set up an 8-byte Opcode Shift 119

Figure 4.22. A Successful 8-Byte Instruction Opcode Shift 119

Figure 4.23. Simplified IA-32 Instruction Format 121

Figure 4.24. Average Instruction Reaches Results 124

Figure 4.25. Overall Distribution of Instruction Sizes 125

Figure 4.26. Snapshot of OllyDbg Showing The Morphed Instructions 127

Figure 4.27. Operation of MOAIS For IDS of MANET 135

Figure 4.28. Pareto Plot of IDS Results 137

Figure 4.29. Antibody Primary and Secondary Response 141

Figure 4.30. Flowchart of REALGO Algorithm 144

Figure 4.31. Generic Signature Data Structure 149

Figure 4.32. The Graph of f6 in One Dimension 152

Figure 4.33. Graph of f1 in One Dimension 152

Figure 4.34. Results of REALGO on function 1 156

Figure 4.35. Results of REALGO on function 8 156

Figure 5.1. Relationships between Objects and Subjects in System 164

Figure 5.2. Online Banking Attack Tree 167

Figure 5.3. Online Banking Protection Tree 168

Figure 6.1. Automated Protection System for Software Applications 179

List of Tables

Table Page

Table 2.1. Risk Calculations for Parent Nodes 8

Table 2.2. Typical Attributes of Non-goat Files 40

Table 2.3. Multiple Ways to Complete Identical Operation. 43

Table 3.1. System Impact Definitions and Numerical Ranges 58

Table 3.2. Rule Set to Propagate Metrics up Attack Tree... 60

Table 3.3. Rule Set To Propagate Metrics Up Protection Tree.................................. 65

Table 4.1. Table of Execution Time Ratio between Morphed and Original Code .. 123

Table 4.2. Table of Metamorphic Engine Performance Results 124

Table 4.3. Table of Observed Occurrences.. 126

Table 4.4. Alphabet of Primitives 132

Table 4.5. Test Functions Used in Testing REALGO 151

Table 4.6. Results of REALGO Algorithm on Eight Test Functions 154

Table 4.7. Results of FES and CES on Eight Test Functions 154

Table 4.8. Student t-tests for REALGO Versus Other Algorithms 155

Table 4.9. Empirically Derived Tuning Parameters for REALGO Algorithm........ 157

List of Symbols

Symbol		Page
I	Impact	8
swp_i	System Weighted Penalty	8
lae_i	Level of Adversary Effort	8
max_r	Maximum Risk	8
E_i	Child Node i	20
$P(G)$	Probability of Goal Node	20
v_i	Node i	20
cc	Can Create Function	30
cr_r	Create Parent	30
cr_c	Create Child	30
c	Child	31
u_n	Parent Subject	31
v	Entity Type	31
N	Number of Parents	31
dom	Domain	31
f_i	Filter Function	31
$risk_{\min}$	Minimum Risk	59

$risk_{normalized}$ Normalized Risk ... 59

$prob_i$ Probability of Node i ... 60

$cost_i$ Cost of Node i ... 60

$impact_i$ Impact of Node i ... 60

n Number of Child Nodes ... 60

fcf Financial Cost Factor ... 66

pcf Performance Cost Factor ... 66

tcf Total Cost Factor ... 66

p Number of Parent Nodes ... 66

P Probability ... 67

C Cost ... 67

R Risk ... 67

μs Microseconds ... 123

l_i Antigen ... 132

$\theta_{\max}$ Maximum Bound for False Positive Detection ... 133

$\xi(\alpha)$ $(1-\alpha)$ Quantile of Normal Distribution ... 133

M_n Number of Detectors with Positive Detection ... 133

n Number of Total Detectors ... 133

w_i Weighting Factor ... 136

F Fitness ... 136

t Time ... 143

$P(x)$	Population at Time x	143
a_i	Antibody	143
$\mathbb{B}$	Binary Number Domain	143
$\vec{v}_i(t)$	Virus Signature	145
p_i	Probability	145
ρ_k	Protected Bit	145
m	Mutation Function	145
λ	Child Population	145
μ	Parent Population	146
ξ_k	RNA Based Cell Indicator	146
ε_i	Threshold Value	146
$s_{(\mu+\lambda)}$	Selection Operator	146
r	Fitness Value for r-Contiguous Bits	150
l	Signature String Length	150
$\varnothing$	Null Set	172

List of Abbreviations

Abbreviation		Page
DoD	Department of Defense	1
SPM	Schematic Protection Model	2
AT&T	American Telephone and Telegraph	5
MIL-STD	Military Standard	5
SDI	Strategic Defense Initiative	5
TLT	Threat Logic Tree	5
SWP	System Weighted Penalty	6
LAE	Level of Adversary Effort	6
NSA	National Security Agency	7
P	Possible	8
I	Impossible	8
BNF	Backus-Naur Form	13
GAC	Goal-Inducing Attack Chain	19
CIF	Component Impact Factor	24
SIF	System Impact Factor	24
EAL	Evaluation Assurance Level	25
CMM	Capability Maturity Model	25
NIST	National Institute of Standards and Technology	25
RTWF	Red Team Work Factor	26
SLE	Single Loss Exposure	27

ALE Annualized Loss Expectancy 27
ROI Return on Investment 27
ROA Return on Attack 27
ESPM Extended Schematic Protection Model 30
TS Subject Type 30
TO Object Types 30
T Set of Entities 30
R Rights 30
BIOS Basic Input/Output 33
MBR Master Boot Record 33
I/O Input/Output 33
API Application Program Interface 33
INT Interrupt 33
ASCII American Standard Code for Information Interchange 39
NOP No Operation 39
DLL Dynamic Link Library 40
PE Portable Executable 40
CPU Central Processing Unit 41
AV Anti-Virus 41
AIS Artificial Immune System 44
BIS Biological Immune System 45
DNA Deoxyribonucleic acid 45
MHC Major Histocompatibility Complex 45

RNA Ribonucleic Acid 45

MANET Mobile Ad-Hoc Network 47

UAV Unmanned Aerial Vehicle........ 47

IDS Intrusion Detection System........ 49

ISA Immune System Agent........ 49

MOAIS Multi-Objective Artificial Immune System 50

K-S Kolmogrov-Smirnov 52

DSR Dynamic Source Routing 52

AODV Ad-Hoc On Demand Distance Vector Routing 52

DSDV Destination-Sequenced Distance Vector Routing........ 52

XML Extensible Markup Language 56

CF Cost Factor 66

FCF Financial Cost Factor 66

PCF Performance Cost Factor........ 66

TCF Total Cost Factor........ 66

DM Decision Maker 66

NCW Network Centric Warfare........ 73

NIPRNET Non-Secure Internet Protocol Router Network........ 74

JRIES Joint Regional Information Exchange System........ 80

DHS Department of Homeland Security 80

HSIN Homeland Security Information Network 80

DDoS Distributed Denial of Service........ 81

DNS Domain Name Service 82

BOTNET	Software Robots	82
IP	Internet Protocol	84
USD	United States Dollar	89
FFIEC	Federal Financial Institutions Examination Council	95
RREQ	Route Request	101
SP	Software Protection	109
HIS	Human Immune System	131
RREP	Route Response	132
RERR	Route Error	132
REALGO	Retrovirus Algorithm	138
CERT	Computer Emergency Response Team	139
RT-RNA	Reverse Transcription Ribonucleic Acid	140
GA	Genetic Algorithm	140
RAM	Random Access Memory	153
FES	Fast Evolutionary Strategies	154
CES	Classical Evolutionary Strategies	155
PIN	Personal Identification Number	161
U	User	161
M	Merchant	161
C	Criminal	161
TN	Transfer Network	161
BE	Bank Employee	161
BS	Bank Server	161

AN	Account Number	161
AID	Account Identification	161
AP	Account Password	161
DCN	Debit Card Number	161
DCP	Debit Card Password	161
AB	Account Balance	161
$	Money	161
BEID	Bank Employee Identification	161
BEP	Bank Employee Password	161
r	Read Right	162
w	Write Right	162
g	Grant Right	162
t	Take Right	162
x	Execute Right	162
c	Copy Right	162
KDF	Key Derivation Function	169

A FRAMEWORK FOR ANALYZING AND MITIGATING THE VULNERABILITIES OF COMPLEX SYSTEMS VIA ATTACK AND PROTECTION TREES

I. Introduction

Critical technology systems are employed daily by various entities ranging from private organizations to the Department of Defense (DoD). Due to the sensitive nature of the systems, security professionals are required to protect information about the systems' inner workings from adversaries. The adversaries have methods, or *attacks,* to break into the systems. Security professionals use techniques, or *protections,* to prevent the adversaries' attacks into the critical systems.

Prior to this research, there were no formal attack or protection plans that discussed where and how attack or protection techniques should be implemented. The placement and use of both types of techniques tended to be that of an art rather than a science. When limited resources are available for protecting critical systems, a tool is needed that allows the security professionals to focus efforts and resources towards the protections that will yield the best performance or return on investment.

1.1 Research Motivation

1.1.1 Goals. The overall goal of this research is to develop a security framework methodology for use in the development of critical systems to determine which protections to implement and where they should be placed within a system. To achieve this goal, a methodology is developed to analyze the security of a system and present recommendations to mitigate the vulnerabilities that were found. Next, this

methodology is shown to be effective using a formal method of analysis. Finally, the methodology is abstracted to multiple application domains to ensure that it is expressive enough to be used on many different types of systems.

1.2 Research Contribution

1.2.1 Framework of Methodology Established. To move the analysis of systems security from an art to a science, a framework for a methodical security analysis and recommendation of solutions is developed. This framework includes a methodical process of creating attack and protection trees, development of metrics and rule sets to propagate the metrics throughout the trees, and tools for the analyst to interface with a decision-maker to select the appropriate protections for the system. Using the novel methods developed in this research, the process of implementing protections into a system is repeatable and unambiguous.

1.2.2 Extension of Schematic Protection Model (SPM). To verify the methodology's effectiveness, the Schematic Protection Model [San88] is used. The basic SPM framework is extended to properly model the constructs of attack and protection trees by implementing a new method of authentication.

1.2.3 Abstraction of Methodology. Analyzing the security of systems is important in many different application domains. To demonstrate the flexibility and value of the methodology developed, security analyses are conducted in many different types of domains. Application domains include an online banking system, a general computer network, a homeland security information sharing network, and a mobile ad hoc network. Through these different application domains, the expressive properties of

the methodology are demonstrated. Basically, any system with an adversary, a protector, and resources that need to be protected can be analyzed with the security methodology developed in this research.

1.3 Research Assumptions

1.3.1 Metrics. To properly analyze the security of a system, certain system dependent metrics used in the analysis of security are required. These metrics may include such things as probabilities of success and costs for attacks and protections. The focus of this research is not on how the metrics are derived but rather on what to measure and how the metrics can be used in the security analysis. The actual measurement or derivation of the specific security metrics is a large enough endeavor to merit a separate research project in itself. As such, the metrics used in the research are representative of actual metrics but are not actual measurements themselves.

1.4 Dissertation Organization

This document is divided into six chapters. Chapter II discusses previous research and provides a background for the development of the methodology. It also covers the previous research in SPM to illustrate the effectiveness of the security methodology. Additionally, background information is presented on research that involves specific protections used in the methodology developed.

Chapter III provides the details of the developed security methodology. It specifically illustrates how attack and protection trees are implemented with metrics to perform a security analysis. Specific application domains are analyzed using the security methodology and discussed in Chapter IV. Chapter IV also presents additional research

that was the impetus for the security framework methodology including using computer viruses to determine protections for software, novel computer virus detection algorithms, exploring security solutions in mobile ad hoc networks, and using metamorphic code to protect proprietary software applications from reverse engineering.

Using the formal methods provided by the SPM, the effectiveness of the security methodology is shown in Chapter V. One of the application domains from Chapter IV is used and SPM is implemented to show that the security methodology does indeed result in a safe system.

Chapter VI provides a brief summary of the research as well as the impact of the new security methodology. Some comments on future research and further development of the methodology are also provided.

II. Background

Attack trees have been around for about 16 years under several different names. Only recently have they been applied to the vulnerabilities within computer networks. Protection trees, which extend attack trees, have been researched very little and as such still have an enormous amount of potential applications. The background presented in this chapter chronicles the development and provides the current state of the art in attack and protection trees as well as the metrics needed for a thorough security analysis.

The development and history of attack trees is presented in Section 2.1. The available research on protection trees and metrics are in Sections 2.2 and 2.3 respectively. The Schematic Protection Model (SPM) is discussed in Section 2.4. Finally, Section 2.5 presents background research on virus protections, human immune system inspired algorithms, and multi-objective systems such as intrusion detection in mobile ad hoc networks. This additional research was instrumental in developing the attack and protection tree methodology.

2.1 Attack Trees

2.1.1 Threat Logic Trees. The seminal paper on attack trees is a publication from AT&T Bell Laboratories [Wei91]. The paper describes a system security engineering process established in response to MIL-STD-1785 that was used during the development of the Strategic Defense Initiative (SDI) for the Department of Defense (DoD). *Threat logic trees (TLT)* are used to decompose high-level potential threats. Weiss' *threat logic trees* are today known as *attack trees* and have both *AND* and *OR*

relationships between nodes. An *AND* node is a node that requires all of its children nodes to be successful to achieve its goal. An *OR* node only requires a single child to be successful to achieve its goal. An attack on a UNIX system is shown in Figure 2.1. The root node of the tree is the overall goal of the attacker; in this case obtaining administrator privileges. This root node is decomposed into child nodes with conditions that lead to the root node. The children nodes are further decomposed until the external attacker actions are determined. Bottom level nodes are known as the *leaf nodes*.

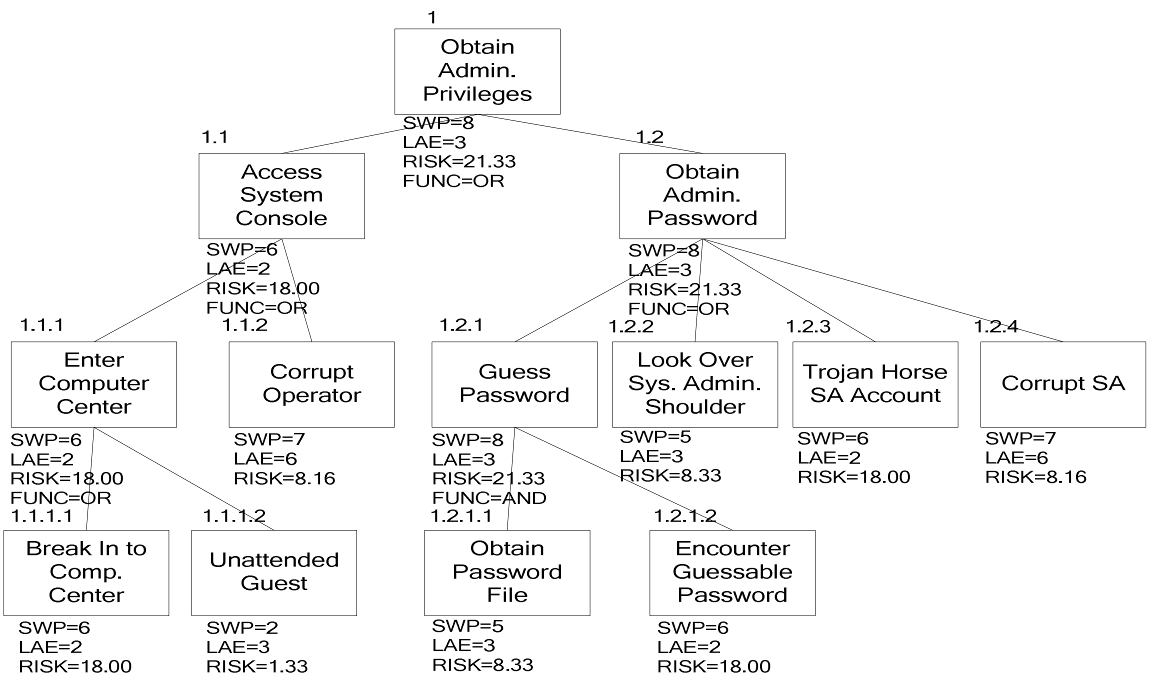

Figure 2.1: Threat Logic Tree Example for UNIX System [Wei91].

Three metrics are associated with each node of a *threat logic tree*. They include *System Weighted Penalty (SWP)*, *Level of Adversary Effort (LAE),* and *Risk.* The SWP is the impact to the system if the threat of the node is successfully accomplished. The LAE is defined as the amount of resources required by the adversary to successfully

accomplish the attack of the specific node. Risk is directly determined by the equation $Risk = SWP^2 / LAE$ [Wei91]. The analyst assigns metric values to the leaf nodes and then determines higher level node metrics using the rules in Table 2.1.

A traditional definition of risk is the 'measure of the probability and severity of adverse effects' [Low76]. This traditionally means risk is calculated by multiplying the impact of an event by the probability of the event occurring. Weiss develops a different definition of risk because he concludes the 'probability of attack is often impossible to estimate' [Wei91]. Since the system often faces an unknown adversary with unknown attributes as well as uncertainty about what may be possible in the future. Weiss' definition of risk is empirically derived from his previous AT&T work in security. The squared SWP term gives extra weight to the impact of severe consequence events. It is presumed an adversary will be willing to spend additional resources for a particularly severe impact to the system.

2.1.2 Attack Tree Term Introduced. Another early paper on a methodology for secure system engineering was written by a working group sponsored by the National Security Agency (NSA) [SaS98]. This paper uses attack trees as one part of a five step process of a methodology for developing a secure system. The attack tree is defined as 'a visualization tool to enumerate and weigh different attacks against a system' [SaS98]. The paper briefly discusses attack trees but does little to develop them.

Schneier continued the NSA work on attack trees and presented them much more extensively in a subsequent paper [Sch99] as a 'formal, methodical way of describing the security of systems' [Sch99]. The attacker's goal is represented as the root node and the

branches from each node represent the sub goals required to achieve the goal of the *parent node*. The sub nodes, or *child nodes,* can either have an *AND* or an *OR* relationship required to achieve the parent node's goal. Figure 2.2 is a simple attack tree based on an attacker trying to break into a physical safe. The author places different metrics on the attack nodes ranging from Boolean values of *possible (P)* or *impossible (IMP) to* a dollar value of the cost of the attack. The metrics are assigned at the leaf nodes and propagate up to the root node. For a more complex attack on an e-mail security program, the author represents the attack tree in outline form as shown in Figure 2.3. Metrics for each node are not included in this format, but they could be added.

Table 2.1: Risk Calculations for Parent Nodes [Wei91].

Risk Calculations		
	AND	OR
SWP	I	$swp_{\max R}$
LAE	$Max_{i=1}^{n} lae_i$	$lae_{\max R}$

where:

I = independently assessed value

swp_{i} = system weighted penalty for child i

lae_{i} = level of adversary effort for child i

n = number of children of the parent

$maxR$ = the child with the maximum associated risk

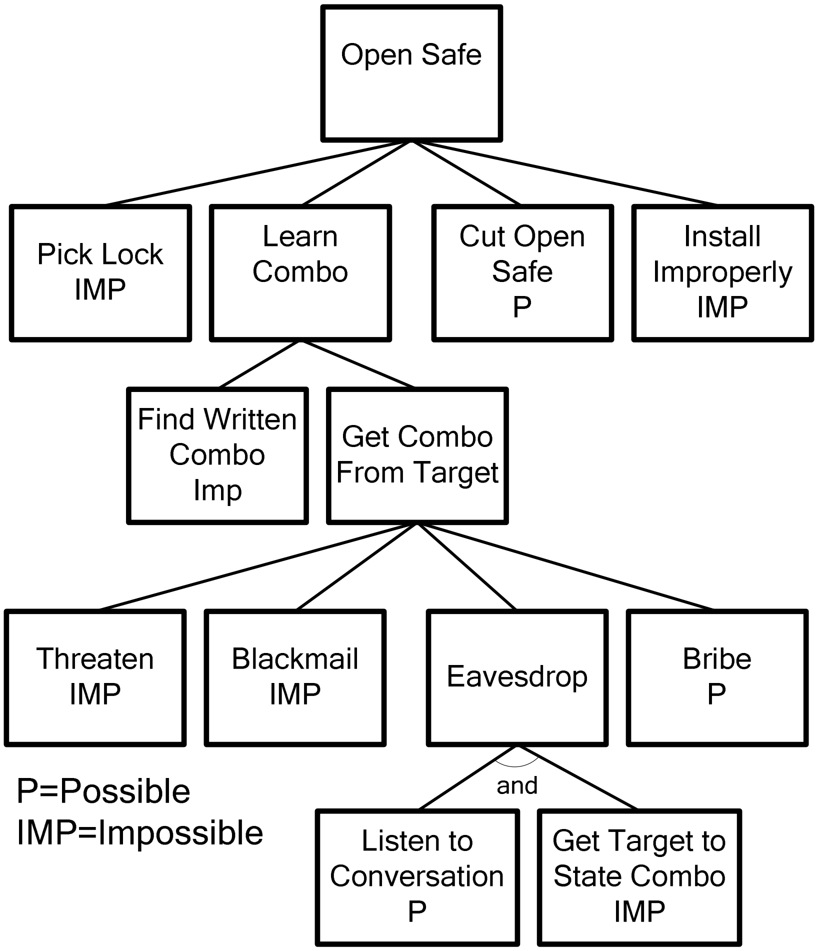

Figure 2.2: Example Attack Tree to Break into a Safe [Sch99].

Constructing an attack tree is an iterative process that takes a certain amount of expertise. Attack goals must be defined and developed into a tree structure. The individual nodes can have metrics assigned to them. These metrics may change over time as technology changes and attacks become easier. The quality of attack trees and the assigning of metrics are based on the experience of the security analyst creating the tree [Sch99].

Schneier lays a good foundation for attack trees but much is left for future development. A more systematic approach, how metrics are determined, and what to do

Goal: Read a message encrypted with PGP. (OR)
1. Decrypt the message itself. (OR)
1.1. Break asymmetric encryption. (OR)
1.1.1. Brute-force break asymmetric encryption. (OR)
1.1.2. Mathematically break asymmetric encryption. (OR)
1.1.2.1 Break RSA. (OR)
1.1.2.2 Factor RSA modulus/calculate ElGamal discrete log.
1.1.3. Cryptanalyze asymmetric encryption.
1.1.3.1. General cryptanalysis of RSA/ElGamal. (OR)
1.1.3.2. Exploiting weaknesses in RSA/ElGamal. (OR)
1.1.3.3. Timing attacks on RSA/ElGamal.
1.2. Break symmetric-key encryption.
1.2.1. Brute-force break symmetric-key encryption. (OR)
1.2.2. Cryptanalysis of symmetric-key encryption.
2. Determine symmetric key used to encrypt the message via other means.
2.1. Fool sender into encrypting message using public key whose private key is known. (OR)
2.1.1. Convince sender that a fake key (with known private key) is the key of the intended recipient.
2.1.2. Convince sender to encrypt using more than one key—the real key of the recipient, and a key whose private key is known.
2.1.3. Have the message encrypted with a different public key in the background, unbeknownst to the sender.
2.2. Have the recipient sign the encrypted symmetric key. (OR)
2.3. Monitor sender's computer memory. (OR)
2.4. Monitor receiver's computer memory. (OR)
2.5. Determine key from pseudorandom number generator. (OR)
2.5.1. Determine state of randseed.bin when message was encrypted. (OR)
2.5.2. Implant software (virus) that deterministically alters the state of randseed.bin. (OR)
2.5.3. Implant software that directly affects the choice of symmetric key.
2.6. Implant virus that exposes the symmetric key.
3. Get recipient to (help) decrypt message. (OR)
3.1. Chosen ciphertext attack on symmetric key. (OR)
3.2. Chosen ciphertext attack on public key. (OR)
3.3. Send the original message to the recipient. (OR)
3.4. Monitor outgoing mail of recipient. (OR)
3.5. Spoof Reply-to: or From: field of original message. (OR)
3.6. Read message after it has been decrypted by recipient.
3.6.1. Copy message off user's hard drive or virtual memory. (OR)
3.6.2. Copy message off backup tapes. (OR)
3.6.3. Monitor network traffic. (OR)
3.6.4. Use electromagnetic snooping techniques to read message as it is displayed on the screen. (OR)
3.6.5. Recover message from printout.
4. Obtain private key of recipient.
4.1. Factor RSA modulus/calculate ElGamal discrete log. (OR)
4.2. Get private key from recipient's key ring. (OR)
4.2.1. Obtain encrypted private key ring. (AND)
4.2.1.1. Copy it from user's hard drive. (OR)
4.2.1.2. Copy it from disk backups. (OR)
4.2.1.3. Monitor network traffic. (OR)
4.2.1.4. Implant virus/worm to expose copy of the encrypted private key.
4.2.2. Decrypt private key.
4.2.2.1. Break IDEA encryption. (OR)
4.2.2.2.1.1. Brute-force break IDEA. (OR)
4.2.2.2.1.2. Cryptanalysis of IDEA.
4.2.2.2. Learn passphrase.
4.2.2.2.1. Monitor keyboard when user types passphrase. (OR)
4.2.2.2.2. Convince user to reveal passphrase. (OR)
4.2.2.2.3. Use keyboard-logging software to record passphrase when typed by user. (OR)
4.2.2.2.4. Guess passphrase.
4.3. Monitor recipient's memory. (OR)
4.4. Implant virus to expose private key.
4.5. Generate insecure public/private key pair for recipient.

Figure 2.3: Text Representation of Attack Tree for an E-mail Security Program [Sch99].

with the attack tree once it is created are some areas where the concept can be extended.

2.1.3 Commercial Implementation of Attack Trees. A commercial product, named SecurITree, uses attack trees in a graphical representation of a system's vulnerabilities [Ame06b]. The program can be integrated with established attack libraries and used within a user's own attack trees. SecurITree is not limited to attacks in the computer domain and can analyze other types of attacks as well. The ability to associate metrics with the nodes is limited to *cost of the attack*, *ability to be detected*, and *technical skill* of the attacker. User defined metrics cannot be added to the model. Some of the metrics are rolled up into higher nodes and some are used to prune the tree under conditions set by the user to find attacks that are more likely based on the assumed resources available to the attacker.

Amenaza, the product's developer, has written several whitepapers about SecureITree. Their 2005 whitepaper [Ame05] lays out the theory behind the current version of the SecureITree software. Three conditions are required for an attacker to carry out an attack. The defender must have *vulnerabilities*, the attacker must have sufficient resources to *exploit* the vulnerabilities, and the attacker must get some *benefit* from the attack. The first condition is determined solely by the defender, the second by a combination of the attacker and the defender, and the third primarily by the attacker. These three conditions are used to attempt to predict the behavior of the attacker and what the defenders impact will be in event of a successful attack.

The attack trees that SecurITree produces are similar to those produced by Schneier. It implements *AND/OR* nodes to produce a tree with a single root node. Leaf

nodes represent the basic attacks and are called *attack stabs*. An example of burglarizing of a house is shown in Figure 2.4. To measure the impact of the attacks, the user can enter *impact indicators* which measure the impact on the defender depending on the type of attack. The impacts are additive to the root node.

To predict which attacks will be used, SecurITree models the assumed capabilities of the attacker. By determining what level of resources an attacker has, the program prunes attacks that are beyond the stated capabilities of the attacker. With the remaining attacks, Amenaza assumes that the attacker will do some type of cost-benefit analysis (not necessarily formal or conscious) and determine which attack will yield the highest return for the lowest cost.

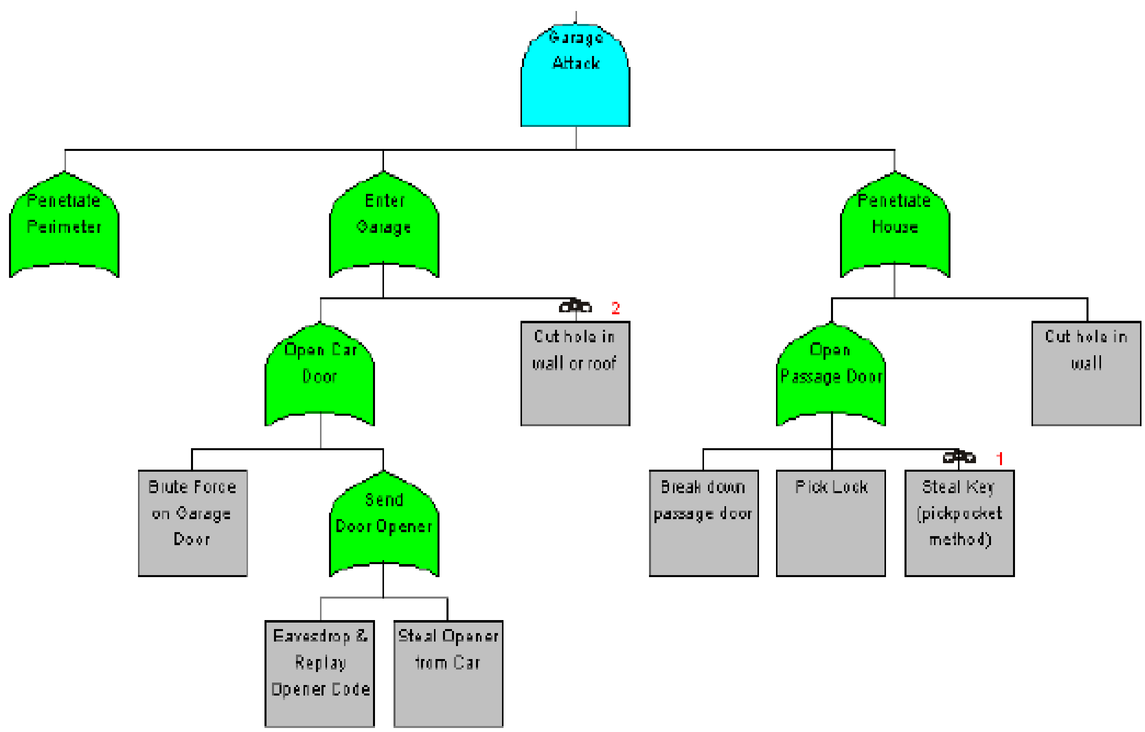

Figure 2.4: Sample Attack Tree Generated by SecurITree [Ame05].

To develop countermeasures, the authors suggest finding portions of the attack tree that are common to multiple scenarios and placing a countermeasure under an *AND* node of the sub tree. Because it is an *AND* node, if the countermeasure cannot easily be defeated, the attack will fail. This appears to be an oversimplified solution. Attack trees may not have common sub trees and the placement of a countermeasure at that particular *AND* node may not be the most cost effective solution. More analysis than SecurITree currently provides needs to be accomplished.

A new version of the SecureITree software [Ame06a] adds nonlinear utility functions to better model the attacker's cost, technical ability, and ability of detection parameters. These parameters are combined to derive an *ease of attack* parameter. The utility functions, however, are created without any mathematical basis. Rather, they are based solely on the opinions of the users. The new version also adds utility functions for the attacker's *benefit* parameter and the defender's *impact* of the attack.

2.1.4 Attack Specification Language. Attacks trees are further extended in a paper using them to model internet attacks [TiL01]. In addition to using the attack tree (Figure 2.5), parameters are added to each node which include descriptive properties, preconditions, post conditions, and sub goals. To represent the parameters, an attack specification language in Backus-Naur Form (BNF) is used as shown in Figure 2.6. The attack specification language is in a form that can be machine processed and used in a notional attack notification system. The paper proposes the language and the notification system but does not develop it into a working system.

2.1.5 Attack Patterns and Profiles. To make attack trees more functional,

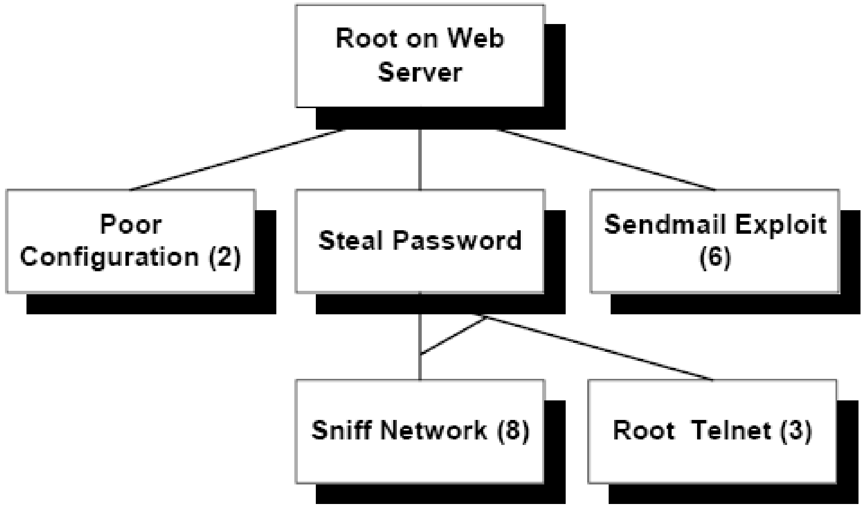

Figure 2.5: Web Server Attack Tree [TiL01].

```
goal ::=  attack name (params)
        {props vars preconds subgoals postconds}
params ::= param (, param)*
param ::=   type ID
props ::=   (prop)*
prop ::=    property  type ID = val  ;
vars ::=      (var)*
var ::=      declare type ID ;
preconds ::= preconditions { orExpr }
orExpr ::=  andExpr ( | andExpr )*
andExpr ::= notExpr ( & notExpr )*
notExpr ::=  grpExpr | ~ grpExpr
grpExpr ::=  atomicRule | ( orExpr )
atomicRule ::=  val boolOp val
boolOp ::=  contains | instanceof | = | > | >=
             | < | <= | !=
subgoals ::= subgoals  { ( andSubgoals )*  }
andSubgoals ::=    (subgoal ( , subgoal )*)
subgoal ::=   name ( args )
args ::=        val ( , val )*
postconds ::=postconditions  {  (stmt )*  }
stmt ::=   name := val ;
val ::=      literal | name
name ::= ID ( . ID )*
literal ::=  version_literal | integer_literal
            | boolean_literal  | date_literal
            | string_literal
type ::=  version | integer | boolean
                 | date | string | name
```

Figure 2.6: Attack Specification Language BNF [TiL01].

Linger and Moore introduced using attack trees with the concepts of *attack patterns* and *attack profiles* [LiM01, MoE01]. An attack pattern describes a single type of attack and can be thought of as a single attack tree. This tree has qualities associated with it which include preconditions that must be present for the attack to occur, the actual steps of the attack, and post conditions if the attack is successful. Attack profiles are a combination of these attack patterns. Different attack profiles model varying capabilities of an attacker such as skills, resources, or type of system access. For attack trees to be useable in real world scenarios, libraries of attack patterns and profiles must be maintained and available to the users [LiM01, MoE01].

Linger and Moore also designed an attack tree refinement process that is depicted in Figure 2.7. This process is a combination of an automated process and expert input. Extending the attack trees and applying the attack patterns continues until the developer is satisfied with the result.

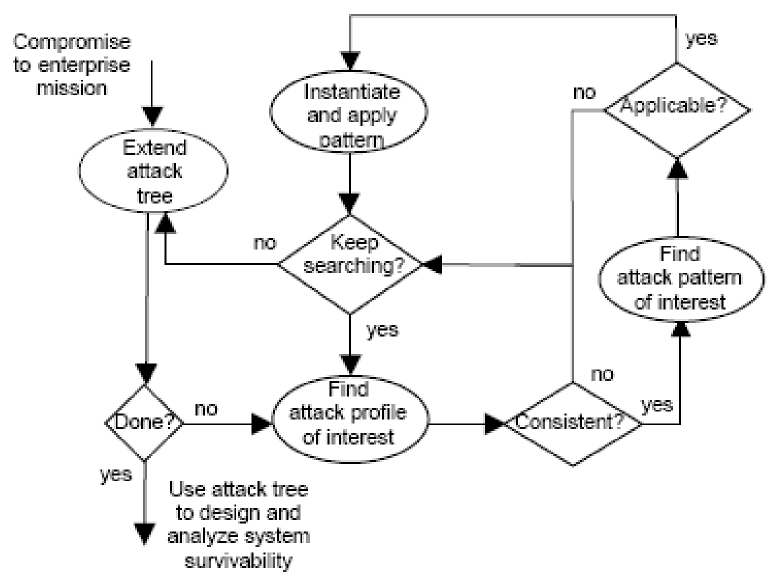

Figure 2.7: Attack Tree Refinement Process [LiM01].

2.1.6 Additional Logic Gates in Attack Trees. Brooke and Paige extend the capabilities of attack trees by adding new types of gates in addition to the standard *AND*/*OR* gates [BrP03]. The logic gates they use are shown in Figure 2.8. The *AND* and *OR* gates are identical in function to other attack tree models. The *PRIORITY AND* gate dictates that child events must occur in a certain order to be successful. The *EXCLUSIVE OR* gate only allows a single event to occur for a successful attack. The *INHIBIT* gate is similar to the *AND* gate except the additional inputs are something from the environment, not an external event. Although the additional types of gates may reduce the total number of gates required in an attack tree, they are not necessary due to the fact they can be modeled using the existing *AND/OR* gate tree constructs.

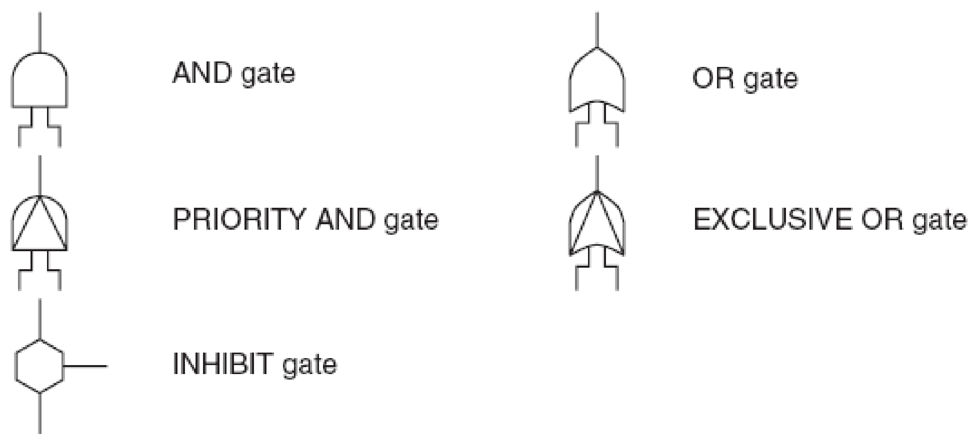

Figure 2.8: Fault Tree Gates [BrP03].

2.1.7 Transformation and Equivalence of Attack Trees. As attack trees grow in complexity, it is important to be able to determine if two trees are equivalent in function. To determine this, there needs to be standard transformation and comparison functions for the attack tree components. Mathematical rigor applied to attack tree design

is the subject of [MaO05]. This work defines attack trees as 3-tuples and nodes as sets. Using these definitions, set operators determine whether two different trees actually contain the same information. Figure 2.9 shows a transformation of an attack tree to an equivalent one with a different representation.

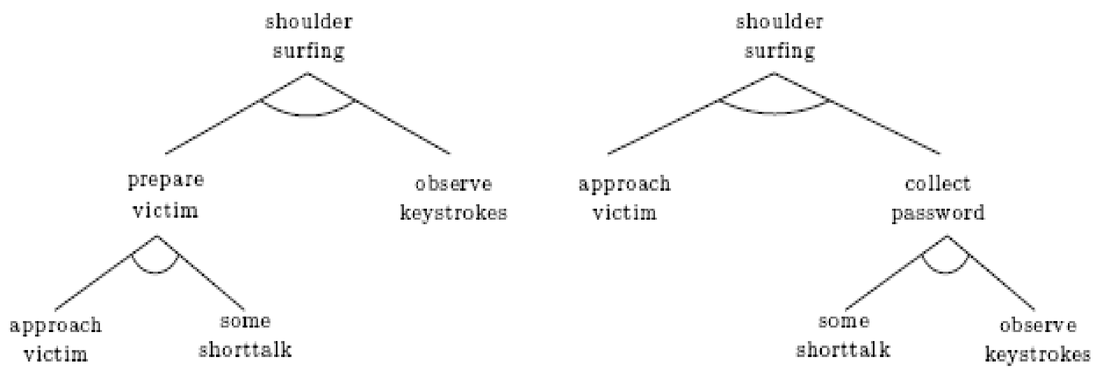

Figure 2.9: Equivalent Attack Trees [MaO05].

To transform trees to different representations, two different rules are used. The first rule, Associativity of Conjunction, states that if a *bundle*, or grouping, of nodes contains a sub-attack of only one sub-bundle, then that sub-attack can be moved up to the same level as the bundle. This is illustrated in Figure 2.10. To complete the attack in Figure 2.10, actions w *AND* (A *AND* B) must be completed. This is equivalent to the attack on the right in which a successful attack is accomplished by completing actions w *AND* A *AND* B. The sub-bundle B is moved up to the top node while keeping the two trees equivalent in function.

The second rule, Distributivity of Conjunction over Disjunction, states that if a bundle contains two or more sub-bundles, then the original bundle can be copied with each sub-bundle being split among the copied bundles. A graphic illustration of this rule

is shown in Figure 2.11. For the first attack to be successful, w *AND* (A *AND* (B *AND* w_1)) must be accomplished. The equivalent tree on the right shows that the attack is successful if w *AND* (A *AND* B) *AND* (A *AND* w_1) are accomplished.

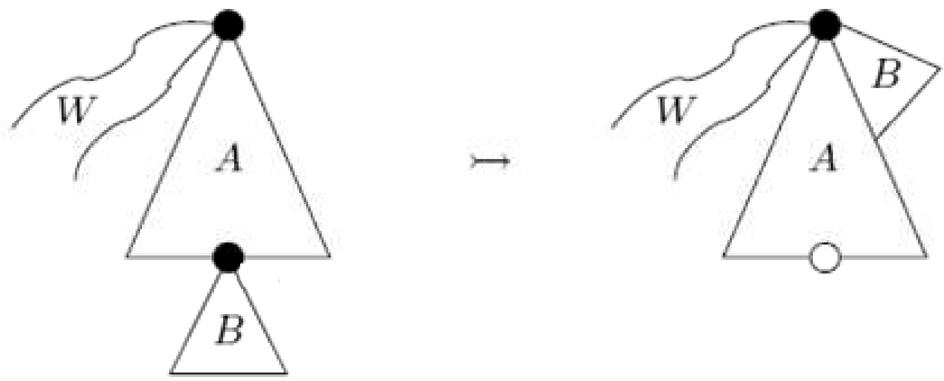

Figure 2.10: Associativity of Conjunction Rule [MaO05].

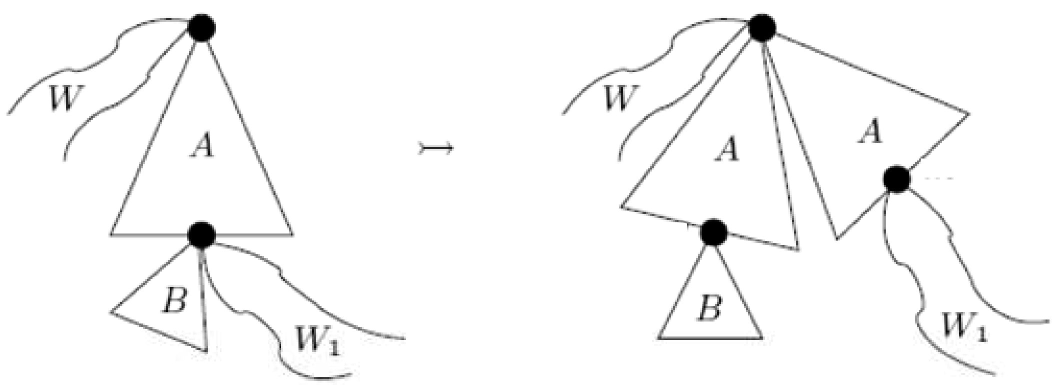

Figure 2.11: Distributivity of Conjunction over Disjunction Rule [MaO05].

2.1.8 Stratified Node Topology. Daley, et al. separates nodes in the tree based on functionality [DaL02]. Three levels of attack nodes are proposed; *event-level, state-level,* and *top-level*. Event level nodes consist of the direct activities of an attacker such as stealing a password. In other attack trees, these types of nodes are known as *leaf nodes*. State-level nodes are intermediate objectives of the attack. An example of a state-

level node would be obtaining a password where sub-nodes might include stealing the password or cracking the password. The top-level nodes represent the attacker's ultimate goal such as obtaining root access on a system. This overall strategy is coined a *Stratified Node Topology*. An example is shown is Figure 2.12. Using this classification, an analyst can combine different attack trees and find common sub trees as well as possibly recognizing a coordinated multi-stage attack.

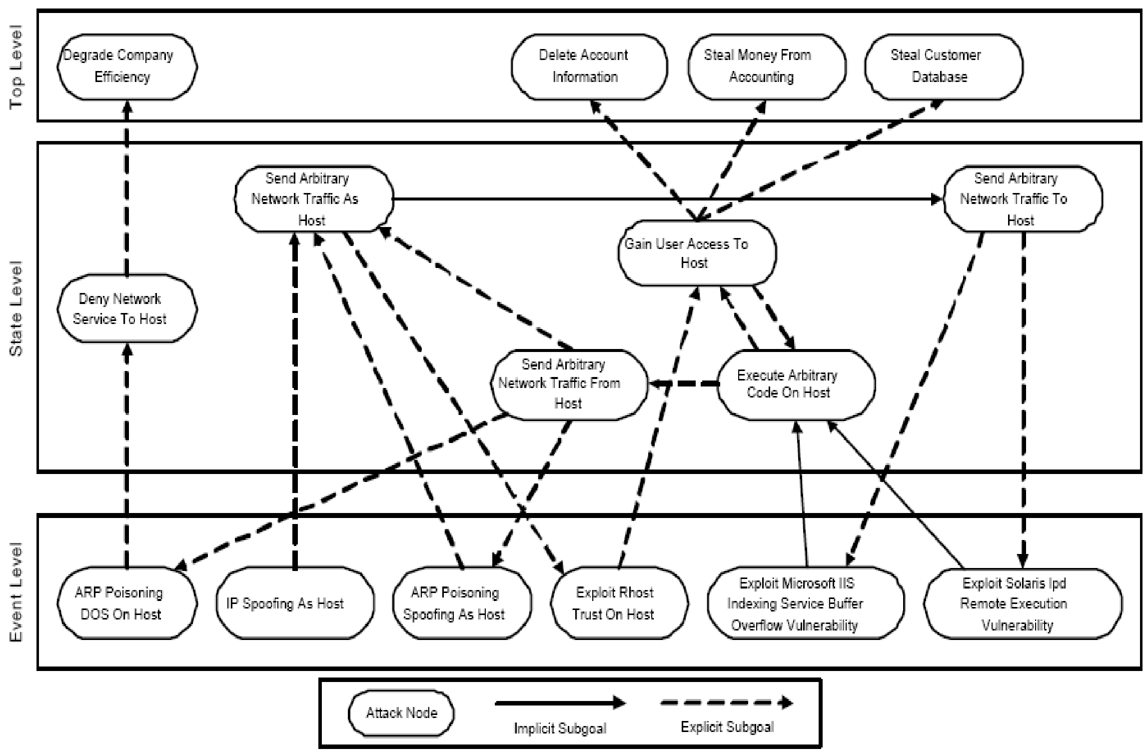

Figure 2.12: Stratified Node Topology [DaL02].

2.1.9 Goal-Inducing Attack Chains. Attack trees are also applicable to multi-stage network attacks [DaH04]. Here, attack trees are constructed by combining *goal-inducing attack chains (GACs)*. Each GAC is a complete branch of an attack tree capable of successfully completing the attack at the root node. In other words, the root node is an

OR node with each child node being the root node of their respective GAC. By comparing different GACs, the minimum cost attack can be found as well as sub attacks that are common to multiple GACs.

To compute probabilities of success for each node in the attack tree, two different formulas are used depending on whether the node is an *AND* or an *OR* node. For *AND* nodes, all sub attacks must be successful so the probabilities are multiplicative as shown in (2.1) [DaH04]. For *OR* nodes, the probabilities are additive and the result is shown in (2.2) [DaH04]. For (2.2) to be true, each child node (E_i) of the node G must be mutually exclusive. If they are not, then (2.2) does not hold as P(G) could be greater than one. If the nodes are not independent then P(G) must be calculated using (2.3).

$$P(E_i) = \prod_{k=0}^{n} P(e_k), i = 0, \ldots, n \tag{2.1}$$

$$P(G) = \sum_{i=0}^{n} P(E_i) \tag{2.2}$$

$$P(G) = \sum_{i=0}^{n} P(E_i) - \sum_{i=1}^{n} \sum_{j=0}^{i-1} P\left(E_i \cap E_j\right), i \neq j \tag{2.3}$$

Consider, for example, the attack tree represented in Figure 2.13. Assume $P(v_0) = 0.15$, $P(v_1) = 0.65$, $P(v_2) = 0.35$, and all nodes are independent. The probability of the left branch of the attack tree using (2.1) is 0.0975. Using (2.2), the probability of obtaining root access is $0.0975 + 0.35 = 0.4475$.

2.1.10 Libraries and Common Knowledge of Vulnerabilities. To properly build an attack tree, the system being analyzed needs to be scanned for flaws. To do that, known taxonomies of flaws can be used. In software systems, a basic taxonomy of security flaws upon which to base an attack tree is helpful [LaB94].

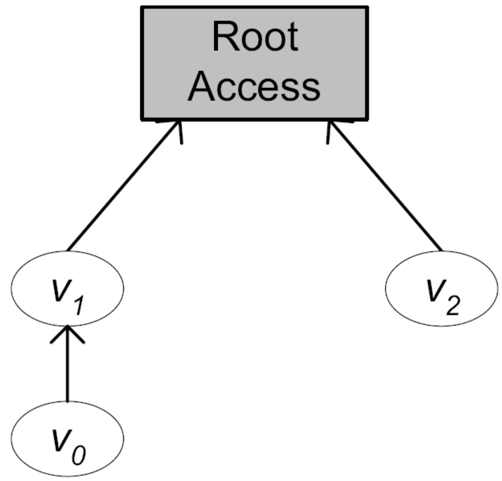

Figure 2.13: Attack Tree Demonstrating Computation of Probabilities in GACs [DaH04].

To gain additional insight into security flaws, a *Wiki Web*-like process is introduced to catalog different types of attacks [StS02]. Their *ATiki attack modeling method* is an attempt to get as many people with any type of attack modeling knowledge together in a forum that makes the exchange of ideas easy. They compare their method to that of open source software development and claim three benefits to this approach including having many experts contribute and share their knowledge, making expert knowledge about typical vulnerabilities available to developers, and making system-specific knowledge available to persons searching for vulnerabilities

2.1.11 Automatic Generation of Attack Trees. The process of automatically generating attack and protection trees is introduced in the field of fault tree analysis [LiR98]. Using finite state machines and failure models (Figure 2.14), faults are injected into the finite state machines to automatically generate fault trees.

2.2 Attack Metrics

2.2.1 Metric Guidelines. Before metrics can be employed, they must be

specifically defined. To obtain a set of useful metrics, a guide to security metrics is developed to explain what needs to be measured and analyzed [Pay01]. A distinction is made between measurements and metrics by basically saying that measurements come from counting things (raw data) and metrics come from analyzing measurements (interpretations of the raw data). Metrics in computer security are essential since it is hard to manage security in an organization if there is not a way to evaluate it.

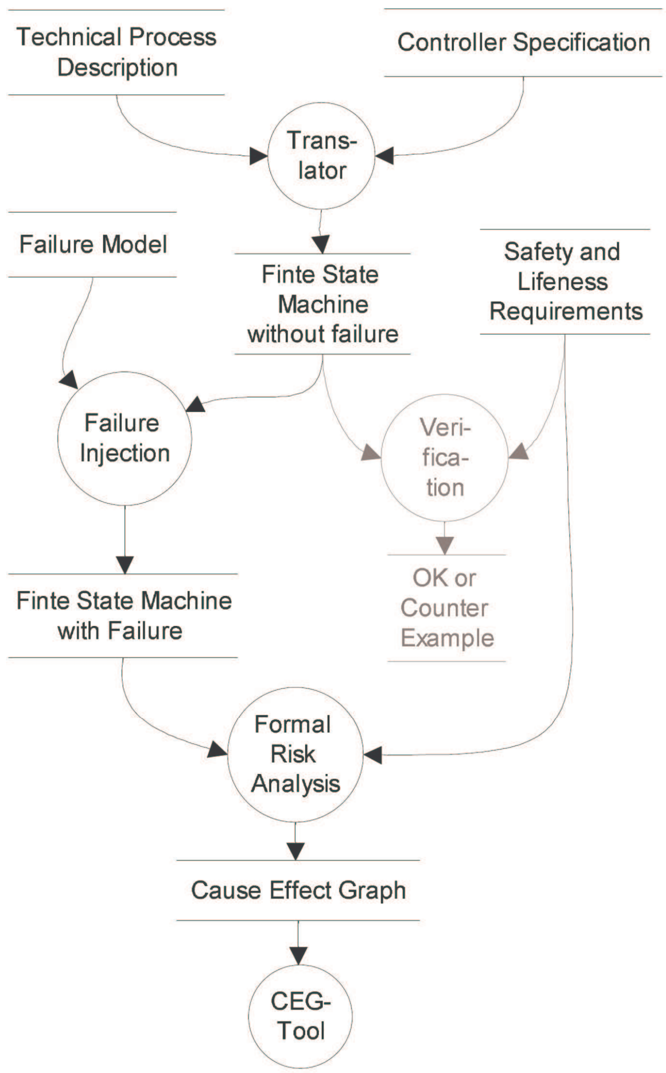

Figure 2.14: Fault Tree Generator [LiR98].

Seven steps are listed for a framework of developing security metrics [Pay01]. They are:

1. Define the metrics program goals(s) and objectives;
2. Decide which metrics to generate;
3. Develop strategies for generating the metrics;
4. Establish benchmarks and targets;
5. Determine how the metrics will be reported;
6. Create an action plan and act on it;
7. Establish a formal program review/refinement cycle.

Schneier also proposed a process for evaluating security [Sch03]. The five steps of this process are:

1. Determine what assets you are trying to protect;
2. Determine the risks to these assets;
3. Evaluate how well the security solution mitigates those risks;
4. Determine what additional risks the security solution causes;
5. Determine the costs and trade-offs the security solution imposes.

While both of these methodologies do not specifically define metrics, they do ensure important questions are answered when evaluating the security of a system. Schneier's process in particular points out the fact that security solutions sometimes create vulnerabilities themselves.

2.2.2 Attack Classifications. In an attempt to determine better software security metrics, malicious software attack classifications are defined in [LaS04]. Rather than

focusing on operating systems, attacks on applications are the focus of this effort. This is significant because research that focuses on attacks at the application level is not nearly as common as those at the network or operating system level. The classification has three elements: location, cause, and impact. An automatic vulnerability scanner is also proposed to test an application for security vulnerabilities. This work is limited in scope in that it proposes a model not an actual process. No results of any type of vulnerability scanning are presented.

2.2.3 Vulnerability Index. Mobile agent research has also been conducted to determine the viability of agents as security monitors. These agents monitor network system states to determine the security of the network or the *vulnerability index* [HaG03]. The *vulnerability analysis engine* uses the vulnerability indexes and calculates metrics for the security state of the system. The process is similar to an intrusion detection system except the security of the network is based on past events. In other words, it does not wait until the intrusion has already taken place. With good indicators it may detect the intrusion before it occurs but after some preliminary attacker actions have occurred.

The *component impact factor (CIF)* is introduced to measure how different parts of the system may be affected if the sensed vulnerabilities are exploited. The CIF is defined differently for different components of the network. For example, a client's CIF is determined by the change in its data transfer rate. A router's CIF is determined by its buffer utilization. A server's CIF is dependent on its connection queue length. A *system impact factor (SIF)* is a measure of how the entire system is affected by the vulnerabilities.

Although interesting ideas and possibly some metrics that can be expanded are presented, the methods introduced really just illustrate what the effects to a network might be if certain *failures* occur. The metrics lend insight into which resources are more important to protect but do not provide useful metrics to determine the vulnerability of a particular component.

2.2.4 Alternatives to Metrics. Another study reviews measurements used in system security and argues that by their very nature, the measurements are hard to come by because they are multi-dimensional which makes them hard to compare [Sch01]. The researcher's solution is to abandon the actual measurement of the systems themselves and instead rate the security of the system based on the processes used to create that system. Some measurements that evaluate the process used in developing software include Evaluation Assurance Levels (EAL) and the Capability Maturity Model (CMM) [Sch01].

The National Institute of Standards and Technology (NIST) has published a Security Metrics Guide for Information Technology Systems [SwB03]. The benefit of the guide is that it lists best practices for administrators of information technology systems. The metrics defined are basically a percentage of systems or users who fulfill certain requirements. Unfortunately, these types of metrics are too rudimentary to be useful for any specific implementation of a system and are too general to use in any type of attack tree.

2.2.5 Empirical Metrics. One approach to security metrics is to measure how much effort a *red team* has to expend to mount a successful attack [ScW00, WoB01]. A red team tests security by trying to break into a system. The effort required by the red

team is a combination of time and money expended by the team and is defined as *red team work factor* (RTWF). Time can be converted into a standard dollar amount to compare different cost units. There are cautions associated with using this type of metric. The first is that the selection of a labor rate is extremely subjective and also has a large impact on the results due to the fact that time is often the largest resource used in an attack [WoB01]. A second caution is that it is hard to determine if variations in metrics are caused by different security situations or variability between the red teams themselves. The final caution is the value obtained from the metric should be the relative differences in the final work factor rather than the absolute values. In other words, using this metric determines which of two attacks is harder for a particular red team rather than determining an attack will always take a certain amount of resources.

A similar study also advocates using *red teaming,* or *penetration testing,* as a rating and ranking system for different security solutions [DoH01]. To use penetration testing as a security metric, sub areas of penetration need to be further developed. These include developing a standard penetration methodology, developing a complete and comprehensive penetration effort, correlating metrics to the penetration effort, and differentiating between systems that are easy to penetrate [DoH01]. Penetration testing can be used to rank similar security solutions although no assertion is made that penetration testing can give accurate metrics for security solutions that protect against dissimilar threats. This research is not based on security at a specific level such as the application, operating system, or network. Instead, it is a general methodology to measure the security of an arbitrary system.

2.3 Protection Trees

To date, there is only one published paper [BiF06] outside of this research which deals with using a protection tree structure in a security analysis. This paper extends attack trees by adding countermeasures to the leaf nodes of the trees. The authors term this extension a *defense tree*. Figure 2.15 shows an example defense tree. The countermeasures in this type of tree only have OR relationships within each class of countermeasures. Economic measures are used as metrics. Some of these include *Single Loss Exposure (SLE), Annualized Loss Expectancy (ALE), Return on Investment (ROI),* and *Return on Attack (ROA).* These economic measures help determine whether a particular protection is worth the monetary investment to implement it based on how much damage it prevents.

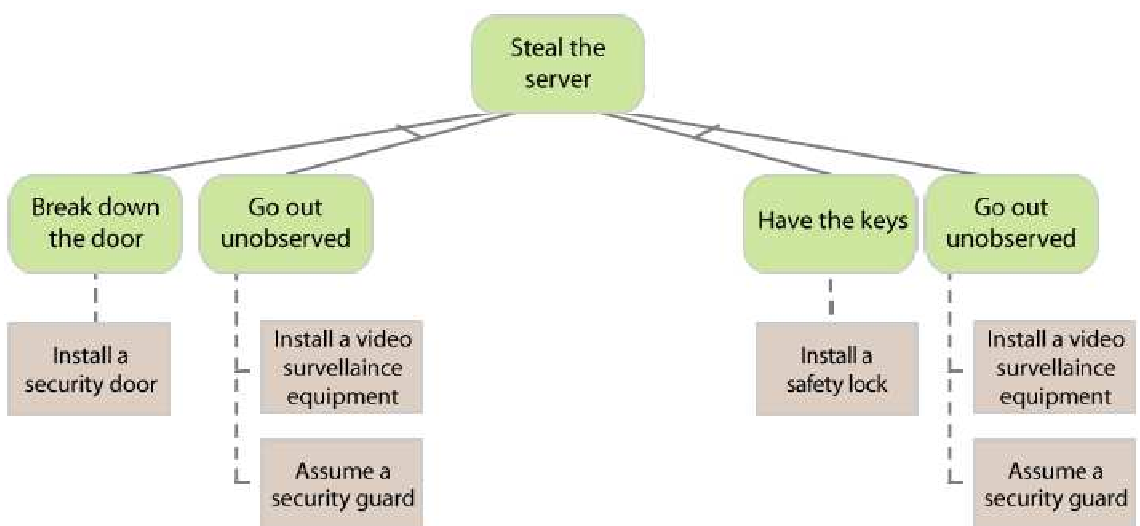

Figure 2.15: An Example of a Defense Tree [BiF06].

SLE is the amount an organization loses from a single attack while ALE is the expected loss from an attack over the course of a year. ROI provides an indicator if a

particular countermeasure results in a positive or negative return based on the expected amount of loss with the countermeasure in place. ROA is how much an attacker expects to gain from a particular attack with countermeasures in place.

2.4 Schematic Protection Model (SPM)

Most security problems decompose into an access control issue. A security violation occurs when an entity who is unauthorized to access some part of the system somehow gains access. To formally specify and analyze a specific security solution, an access control model is used. In this research, the SPM is used because of its ability to analyze security based on the propagation of access rights.

2.4.1 Safety Question. As stated above, the key to determining if a security violation can occur is by determining if a given entity can obtain a particular right to access some part of the system in question. In most computer systems, it is important to ensure only users who are authorized to possess certain rights over objects actually do. If a user obtains rights he is unauthorized to have over an object, a *leak* has occurred. For example, if a user not authorized to have a read right over a file obtains that right, then the read right to that file has been leaked. When a system leaks a right *r*, it is said to be *unsafe* with respect to right *r* [Bis03]. To determine if a system is safe, an analysis has to determine that rights cannot be leaked to users not authorized to possess them. This is known as the *safety question*.

2.4.2 Schematic Protection Model. To show that implementing attack and protection trees to determine the security scheme of a system results in a safe system, the system needs to be modeled and analyzed. The model used to analyze the system is the

Schematic Protection Model [San88]. Although full details of the SPM can be found in [AmS90, AmS91, San88, San89], the fundamental properties of SPM are covered below.

SPM is based on the concept of a *protection type*. Before the protection type can be explained, a few terms need to be defined. In any system there are two types of rights; *control rights* and *inert rights*. Control rights alter the protection state of the system while inert rights do not. In SPM, control rights are used to establish links between entities. The entities in the system have a *protection type.* The protection type determines how control rights affect the entity. Once a protection type is defined for an entity, it does not change.

For a right to be transferred between entities there must be a *copy flag* associated with the right. The transfer of a right also depends on two relationships between entities; the *link predicate* and *filter function*. The link predicate determines if the entities are logically connected while the filter function specifies which rights may be transferred between the entities.

A single right is called a *ticket*. A ticket lists the entity that the right applies to and the right's symbol. For example, the ticket *X/r* gives the entity possessing it *r* rights over entity *X*. An entity's set of tickets is the *domain* of the entity and contains all of the rights the entity holds to other entities in the system.

Entities in SPM are created using the specified *create* rules. For every pair of protection types there is a corresponding create rule. The rule specifies which tickets are placed in the creating and created entities domains when the create rule is exercised. To keep the analysis tractable, create rules ensure the system remains *acyclic* with respect to

entity types. This eliminates any cases of recursion in the system. If the create rules do not prevent cyclic entities, the safety problem has been shown to be undecidable [San92].

To answer the safety question for a particular system, the system is first defined using SPM and is represented in its *initial state*. Subjects are created until a *maximal state* is achieved. This is the state in which adding an additional subject does not affect the protection scheme. A maximal state is not necessarily unique [San88]. Once a maximal state is determined, a *maximal flow* can be determined which shows which entities can obtain which tickets. If the system is safe, only authorized entities can obtain the appropriate tickets. If the system is unsafe, leakage of rights can occur which means entities can obtain unauthorized tickets.

A short example follows. The actual scheme represented is that of the Extended SPM or ESPM where the only difference is that created entities can have multiple parent entities [AmS91]. The SPM schema has

(1) Disjoint sets of subject types TS and object types TO. Let $T = TS \cup TO$ where T is the set of all entities in the system.

(2) A set of rights R. The set of possible tickets is thereby $T \times R$.

(3) A can-create function: $cc: TS \times TS \times \cdots \times TS \rightarrow 2^T$ where 2^T is the power set of the combinations of all types. This means there is a create function defined for every type of subject creating every other type of subject.

(4) Create rules of the form: $cr_{p_i}(u_1, u_2, \ldots, u_N, v) = c/R_3 \cup p_i/R_2^i$ for $i = 1..N$ and

$$cr_c(u_1, u_2, \ldots, u_N, v) = c/R_3 \cup p_1/R_4^1 \cup p_2/R_4^2 \cdots \cup p_N/R_4^N$$

where p_i is the *ith* parent and c is the child, u_n are the parent subjects, v is the created entity type, N is the number of parents, and R are the rights. When an entity is created, the create rules specify which rights the parents of the entity possess and which rights the created entity possess after the create operation.

(5) A collection of link predicates $\{link_i\}$ which define the logical connections between entities.

(6) A filter function $f_i : TS \times TS \rightarrow 2^{T \times R}$ for each predicate $link_i$ which specify the rights that can be transferred between subjects where $2^{T \times R}$ is the power set of all types and rights.

A specific instance of this schema follows.

(1) Bob, Alice are of protection type a; Eve is type b

(2) $x \in R$; x is a control right.

(3) $cc(a,a) = b$; entities of type a (Alice and Bob) can jointly create entities of type b (Eve).

(4) $cr_{Alice}(a,a,b) = cr_{Bob}(a,a,b) = \varnothing$; Alice and Bob do not get any rights when they create a subject of type b

$cr_{Eve}(a,a,b) = \{Alice / x, Bob / x\}$; When Eve is created by Alice and Bob she has the x right over Alice and Bob (known as a ticket).

(5) $link(Alice, Bob) = Alice / x \in dom(Bob)$; a link exists between Alice and Bob if Bob has the right x for Alice in his domain (*dom*)

(6) $f(Bob, Alice) = \{Alice / x\}$; the x right for Alice may be transferred between Bob and Alice

2.5 Background in Application Areas

2.5.1 Virus Protections. Many of today's computer viruses use advanced techniques to protect themselves from discovery and reverse engineering to extend infection and propagation times. These techniques are often very sophisticated and powerful, developed by people who understand the underlying computer architectures and operating systems at very detailed levels.

In 1995, the first year that Virus Bulletin kept statistics; there were approximately 2,300 instances of computer viruses reported [Vir95]. By 2004, that number had steadily increased to nearly 2.5 million [Vir04]. Although a large part of that number is due to minor variants of a few viruses, virus proliferation has become an ever increasing problem for computer systems today.

This area of the research determines how protections that viruses use can be used to protect software. The classifications of the different protections are discussed below and the applications of the protections are discussed in Chapter IV.

2.5.1.1 Classification of Protections. As mentioned before, a primary goal of viruses is to remain undetected in their host computer systems. Should they be detected, they try to make reverse engineering as difficult as possible by employing a variety of techniques and often combining multiple methods. The techniques include memory-resident tunneling, encryption, stealth, retrovirus, and armoring.

2.5.1.1.1 Tunneling. When computer systems boot up, after loading and running

the Basic Input/Output System (BIOS), the system looks for the Master Boot Record (MBR). This is the record on the hard drive that tells the computer how to load the operating system. Viruses sometimes attempt to become the first program the computer loads and thereby remain undetectable by bypassing items such anti-virus applications. If the virus is loaded before the virus scanner, then it simply instructs the computer to never load the anti-virus program. It can also "hook" other system functions such as directory listings that might detect the presence of a virus and return to the user only the information the virus thinks will keep it hidden [Szo05]. Dark_Avenger.1800.A also known as Eddie employs memory scanning to find the original handler to hook. Yankee_Doodle uses tracing (debugging tools) to find the interrupt (INT) hook. Nokernal uses direct access to the disk via Input/Output (I/O) ports to bypass interrupts and Application Program Interfaces (API).

2.5.1.1.2 Encryption Encryption is one of the most widely used protections for viruses. Because most scanners use signature detection to find viruses, encryption with multiple keys can "hide" the signature of a virus so it can remain undetected. Encryption also makes the program harder to understand for the virus researcher. Some examples of viruses employing encryption include W32/Blaster, W95/Fix2001, and ABR-1171.COM. W32/Blaster uses compressed and obfuscated code. W95/Fix2001 uses encryption to hide e-mail addresses it sends local information to. ABR-1171.COM uses a decryptor to decrypt the virus body prior to execution. Fortunately, most virus encryption methods are quite simple using a simple command such as `xor` [Szo05]. With this type of encryption, the researchers can usually find the key using brute-force

tactics.

An example listing of assembly code for the ABR-1171.COM virus follows [Bla96]. The code is run through a debugger to see how it decrypts itself.

```
-u100
 14CF:0100 E99004     JMP     0593     ; Jumpt to the address 593h
 14CF:0103 3F         AAS              ; The rest of the code, as you
 14CF:0104 95         XCHG    BP,AX    ; can see, is encrypted and has
 14CF:0105 D7         XLAT             ; no sense at all %-)
 14CF:0106 29A6C13F   SUB     [BP+3FC1],SP
 14CF:010A 8FD7       POP     DI
 14CF:010C 29A2C13F   SUB     [BP+SI+3FC1],SP
 14CF:0110 16         PUSH    SS
 14CF:0111 D7         XLAT
 14CF:0112 7F6E       JG      0182
```

The rest of the decrypted code starts at address 593h.

```
-u593
 14CF:0593 BE0001     MOV     SI,0100  ; 100h address of actual jmp
                                       ; 593h
 14CF:0596 56         PUSH    SI       ; Save SI
 14CF:0597 B94A02     MOV     CX,024A  ; Number of bytes to decrypt
 14CF:059A C70429D8   MOV     WORD PTR [SI],D829  ; d829 at 100h
 14CF:059E C64402C1   MOV     BYTE PTR [SI+02],C1 ; c1h at 102h
 14CF:05A2 8134C1D7   XOR     WORD PTR [SI],D7C1  ; d7c1h at SI. At
                                                  ;100h there is a
                                                  ;call c212h
```

At address 0x199h is the partially decrypted header where instead of a `jmp 593h`, there is a new instruction, `call c212h`.

```
-u100
14CF:0100 E80FC1     CALL    C212  ; 100h address right now
14CF:0103 3F         AAS
```

Continuing the decryption...

```
14CF:05A6 46         INC     SI   ; Increase SI, to get 101h
14CF:05A7 46         INC     SI   ; Increase SI, to get 102h
14CF:05A8 E2F8       LOOP    05A2 ; Decrypt 24ah bytes which is
                                  ;in CX
14CF:05AA 31F6       XOR     SI,SI; Get a 0 at SI
14CF:05AC 31C9       XOR     CX,CX; Get a 0 at CX
14CF:05AE C3         RET
14CF:05AF 0000       ADD     [BX+SI],AL
```

The disassembled code starting at 100h looks like this after decryption. The virus is now ready for execution.

```
-u100
14CF:0100 E80F00     CALL    0112
14CF:0103 E85400     CALL    015A
14CF:0106 E87100     CALL    017A
14CF:0109 E84E00     CALL    015A
14CF:010C E87500     CALL    0184
14CF:010F E8D700     CALL    01E9
14CF:0112 BEB904     MOV     SI,04B9
14CF:0115 8B1C       MOV     BX,[SI]
14CF:0117 0BDB       OR      BX,BX
14CF:0119 743E       JZ      0159
14CF:011B B8DD34     MOV     AX,34DD
14CF:011E BA1200     MOV     DX,0012
```

2.5.1.1.3 Stealth. Stealth viruses try to hide from the user by intercepting system calls that would normally indicate the presence of a virus [KaS05]. Stealth viruses are characteristically active in memory and manipulate data that the user requests to hide their appearance [Szo05]. These types of viruses perform a "man-in-the-middle" attack between user applications and the operating system (or the hardware itself). For example, an infected program likely is larger than the uninfected version. When a user queries for the size of the file, the virus in memory manipulates the actual size and returns a value that effectively hides the virus. Four types of stealth covered here are: semi-stealth (or directory stealth), read stealth, cluster- and sector-level file stealth, and hardware stealth. Stealth techniques are also commonly found in rootkit technology [Erd03].

2.5.1.1.4 Semi-Stealth. This type of stealth hides the change in file size when a user lists the directory contents but the changed content is still accessible (Szor 2005). Two examples are Vienna which sets the time/date stamp as a marker so that the

virus knows to manipulate file size and W32/Cabanas which is described in the following code excerpt [Jac97a]:

```
;at this point, calculate virus checksum to make sure file is really
;infected. If its infected then return original size of host previous
;to infection and store it in the WIN32_FIND_DATA structure (stealth).

    mov     eax,[edi.NT_OptionalHeader.OH_CheckSum] ;get checksum field
    push    eax
    sub     al,2Dh          ;calculate virus checksum to make sure file
                            ;is really infected
    xor     ah,al
    mov     al,[edi.NT_FileHeader.FH_TimeDateStamp.hiw.hib]
    xor     ah,byte ptr [edi.NT_OptionalHeader.OH_CheckSum.hiw]
    and     al,11111100b
    xor     ah,al
    mov     [ebp + uni_or_ansi - ebp_num],ah
    inc     ah
    pop     eax
    jnz     go_esi
    xor     eax,0B2FD26A3h xor 68000000h
    xor     eax,[edi.NT_FileHeader.FH_TimeDateStamp]
    and     eax,03FFFFFFh
    cmp     eax,[ebx.WFD_nFileSizeLow]
    jnc     go_esi
    mov     [ebx.WFD_nFileSizeLow],eax     ;return original file size
go_esi: inc     esi                        ;set "already infected" mark
```

2.5.1.1.5 Read Stealth. This type of virus intercepts access to the first sector of the disk (boot) and replaces it with bogus information [Fse05]. The user cannot read the correct information. Three examples of read stealth viruses are Brain, Stoned, and Michelangelo.

2.5.1.1.6 Cluster- and Sector-Level File Stealth. The virus displaces program code in the host program and places it in the cluster slack (empty) space. This method keeps the file size unchanged meaning the virus does not have to intercept any system calls to the user. This method is much harder to employ and limits the size of the viruses depending on the available empty space in the host programs. A virus which employs this technique is Number_of_the_beast.

2.5.1.1.7 Hardware-Level Stealth. A virus achieves hardware-level stealth by interfacing with the hardware at the interrupt level. By hooking the appropriate interrupts, the virus is able to intercept any reads of the disk that may contain traces of its existence. The Strange virus uses this technique [Szo05]. When another program attempts to access a virus-infected sector, Strange relays the original, uninfected sector, which effectively hides the virus from the requesting application.

2.5.1.1.8 Armoring. This virus category makes reverse-engineering very difficult. The goal is to increase the time needed by virus researchers to understand them while they propagate. Virus writers understand that once the researchers understand the operation of a virus, they can release a signature and patch for it. To hinder this, virus writers armor their viruses using anti-debugging and anti-heuristic techniques for both static and dynamic analysis. Malware developers prevent their programs from infecting goat (or bait) files by using anti-goat (also known as anti-bait) techniques. Viruses that infect such files simplify the virus researcher's task.

2.5.1.1.9 Anti-debugging. The basic goal of anti-debugging strategies is to disable necessary features to make the debugger fail. To detect debuggers, malware applications use several different techniques. Hooking INT 1 (single step interrupt) and INT 3 (breakpoint interrupt) are common anti-debugging approaches as most debuggers commonly use these interrupts [Szo05]. Viruses also scan for registry keys that debuggers commonly use and may scan memory itself for debuggers. Due to the hardware support required for debugging, anti-debug features tend to be very platform specific.

The example below checks to see if the keyboard is being used (a sign of a debugger). If it is, the virus will not decrypt the code [Lor97].

```
mov      ax, 0b00h   ; get keyboard status
int      21h         ; this returns al with FFh or 0h
dec      al          ; decrement al to have a
jns      not_good    ; signed number
jmp      decryptor   ; which will lead to the decryptor.
                     ;not_good:
```

2.5.1.1.10 Anti-heuristics. In general, a common anti-heuristic strategy adopted by malware is to confuse or mislead heuristic scanners. Certain malware programs try to confuse heuristic scanners as they attempt to determine if a particular section of code is malicious. Some approaches used in malware include recalculating checksums and not using common API function names [Szo05].

Heuristics generally consist of two phases: a static phase, which looks at the structure of the code to detect a virus, and a dynamic phase, which looks at the behavior. Encrypting or packing code many times can fool static analyzers. Some viruses even pack the code at many levels so the analyzer does not know how many levels it needs to unpack.

Viruses attempt to disable dynamic heuristic analyzers by executing commands that may cause them to fail. They may throw an exception to a handler to give control to another part of the code. They might use an undocumented instruction. They could use extremely long loops. In all of these cases, an emulator will probably fail thus allowing the virus to escape detection with a dynamic heuristic scanner.

Viruses that employ static anti-heuristic techniques include W32/Explorezip, W95/SK, and W32/IKX. W32/Explorezip fools scanners using packing. W95/SK fools analyzers looking for a writable flag because it does not set any sections to writable.

W32/IKX recalculates the code section size in the header to ensure it matches with the virus attached.

Examples of dynamic anti-heuristic viruses are W95/Champ.5447.B, W95/Vulcano, and W32/Gobi. W95/Champ.5447.B sets up an exception handler which the virus uses to pass control to another part of virus code. W95/Vulcano implements an undocumented instruction (SALC) that confuses some emulators. W32/Gobi employs long loops (i.e. 40 million) to generate a decryption key can make emulation extremely slow.

2.5.1.1.11 Anti-goat Viruses. Researchers use goat programs (also known as bait programs) to aid in reverse engineering. By providing a "host" file with a simple, known format, the virus infection points become trivial to spot. A typical goat program might contain all 0x41H (ASCII A) or No Operations (NOPS) making any changes the virus makes during infection obvious. To avoid falling "victim" to such a simple analysis, a virus refrains from infecting goat files. Anti-goat viruses run checks on the host programs to see if they meet certain conditions and then make a decision on whether to infect or not. They might check attributes such as size, file type, and file name.

Below in Table 2.2 are common attributes and typical values that malware evaluates to determine if infection is appropriate [Sna04]. If the conditions do not hold then the virus assumes that the target host is a goat file and will not infect it. Viruses make trade-offs between behavior that is too cautious and too risky with respect to infection criteria. Not considering enough attributes could lead to infecting obvious goat files, while contemplating too many attributes could result in a virus that is too timid to

propagate [Szo05].

Table 2.2: Typical Attributes of Non-goat Files [Sna04].

Attribute	Typical Values
Size	greater than 40 KB
Imported APIs	check import table for more than 30
Used Dynamic Link Libraries (DLL)	more than 5
Data Size	check offset 0x20h in Portable Executable (PE) header (should be greater than 12K)
Code Size	check offset 0x2ch in PE header (should be greater than 5K)
Resources	check .data section for size greater than 800B
Repeating Items	check file name similarities, code sections; one technique is to check first 2-3K bytes and compute a checksum and compare files before infecting

The following is an example of checking file attributes before infecting from the Jacky.1440 virus [Jac97b]:

```
Process_File:   ;check file and infect it

    lea     edx,[ebx.WFD_szFileName]
    call    Open&MapFile                  ;open and map file
    jecxz   Find_Next
    xor     eax,eax
    cmp     [ebx.WFD_nFileSizeHigh],eax    ;skip filez too large (>1GB)
    jnz     Close_File
    add     eax,[ebx.WFD_nFileSizeLow]
    js      Close_File
    add     eax,-80h                      ;skip filez too short
    jnc     Close_File
    call    Check_PE_sign                 ;it has to be a PE file
    jnz     Close_File
    test    ah,IMAGE_FILE_DLL shr 8       ;can't have DLL bit
    jnz     Close_File
    xor     ecx,ecx
    mov     eax,[ebx.WFD_nFileSizeLow]    ;check if file is infected
    mov     cl,size_pad
    cdq
    div     ecx
    mov     esi,edx ;esi == 0, file already infected or not infectable
                    ;esi != 0, file not infected, i.e. infect it!
Close_File:
```

```
call    Close&UnmapFile                ;close and unmap file
mov     ecx,esi
jecxz   Find_Next                      ;jump and find next file
call    Infect                         ;infect file
```

2.5.1.1.12 Anti-emulation Armoring. Two common anti-emulation strategies used in malware are confusion and "resource consumption" (i.e., denial of service). Multi-threaded malware can be more difficult to examine than a program that has a single execution thread. Executing within a hierarchically structured exception handling environment can be confusing for an attacker and an emulator. Using coprocessor functions and undocumented central processing unit (CPU) instructions can fool emulators and cause them to lose their state.

2.5.1.1.13 Retrovirus. Viruses are typically on the defensive against anti-virus scanners. A retrovirus takes an offensive position against the anti-virus programs. These types of viruses specifically try to bypass or hinder anti-virus programs, firewalls, or other security programs. Typical methods include disabling the scanners, deleting or changing the signature databases (Gollum and IDEA.6155), or simply not allowing updates to the signature databases by blocking internet access to the anti-virus (AV) update sites (W32/Mydoom and W95/MTX) [Szo05].

An example from the Gollum virus shows how it deletes the anti-virus signature databases for many leading anti-virus programs [Gri97]:

```
;Delete ANTI-VIR.DAT
mov esi,OFFSET32 CheckSum_File_00
call     Delete_File
;Delete CHKLIST.TAV
mov     esi,OFFSET32 CheckSum_File_01
call    Delete_File
;Delete CHKLIST.MS
mov     esi,OFFSET32 CheckSum_File_02
call    Delete_File
;Delete AVP.CRC
```

```
mov    esi,OFFSET32 CheckSum_File_03
call    Delete_File
;Delete IVB.NTZ
mov    esi,OFFSET32 CheckSum_File_04
call    Delete_File
```

2.5.1.1.14 Morphing Viruses. Currently the biggest virus detection challenge is the presence of viruses that change or "morph" from generation to generation. Because the virus changes, it makes it nearly impossible to use signature-based detection. It also makes it very hard to analyze the code because every instance can be different. There are two different levels of morphing viruses discussed below. They are polymorphic and metamorphic.

2.5.1.1.15 Polymorphic Viruses. Polymorphic viruses can generate a large number—possibly millions—of variants by mutating their decrypting sections to fool virus scanners. This technique makes it extremely difficult and impractical to use a signature-based scanner to detect the virus, because the most static portion of the virus is usually the virus body decryptor. By adopting polymorphic tactics, detectors cannot rely on the signature of the decryptor alone. The Spanska.4250 virus uses a probabilistic method for determining how it encrypts and decrypts itself [Spa97].

```
mov     ax,100
call    aleatoire
cmp     ax,20             ;20% chances for a XOR encryption
ja      evite_suite
jmp     cryptage_xor
evite_suite:
cmp     ax,40             ;20% chances for a ADD/SUB encryption
jb      cryptage_add
cmp     ax,55             ;15% chances for a ROL/ROR encryption
jb      cryptage_rol
cmp     ax,70             ;15% chances for a INC/DEC encryption
jb      cryptage_inc
cmp     ax,85             ;15% chances for a NOT encryption
jb      cryptage_not
                          ;15% chances for a NEG encryption
```

2.5.1.1.16 Metamorphic Viruses. Metamorphic viruses generate an even larger number of variants by using more advanced tactics such as subroutine reordering, instruction substitution, and instruction reordering. Metamorphic viruses mutate their instructions to change the "shape" of the virus while keeping the same functionality. Malware metamorphism occurs during infection and propagation.

Table 2.3 shows several ways to perform the same task. All of the following opcode combinations put 1000h into the bx register. To exchange larger size opcodes with smaller equivalent ones, the metamorphic engine pads the excess space with nops or junk instructions [Lor97]. This technique obfuscates the code as well.

Table 2.3: Multiple Ways to Complete Identical Operation [Lor97].

Instruction	OpCodes	Total bytes
`mov bx,1000h`	B8 00 10	3
`xor bx,bx` `or bx,1000h`	33 DB 81 CB 00 10	6
`push 1000h` `pop bx`	68 00 10 5B	4
`sub bx,bx` `xor bx,1000h`	2B DB 81 F3 00 10	6
`mov bx,1000h xor 2222h` `xor bx,2222h`	BB 22 32 81 F3 22 22	7

There are many different ways to generate junk instructions. A list of the most common follows:

```
90h = nop
f8h = clc
f9h = stc
fah = cli
fbh = sti
fch = cld
dch = std
```

Although usable as nops, the last two can also have other results when using the ci, di, and cx registers. Other garbage instructions could include manipulating registers

that are no longer necessary and performing meaningless calculations with other registers while storing the results in a garbage register.

The Spanska.4250 virus takes a different approach. It carries around several mutations within its code and then chooses one to send forward. This method is a quick and easy way to create a metamorphic virus but it has the drawback of making the virus very large. A sample from the code shows it has nine ways of doing the following instruction [Spa97]. Three ways are shown for illustrative purposes.

```
;7/ mov di, si in 4 bytes (without CX, SI)

_mutation7:

mov di, si
nop
nop

sub di, di
xor di, si

xchg si, ax
xchg ax, di
mov si, di
```

2.5.1.2 Summary of Virus Protections. Malicious software developers continue to challenge the protection community with advanced techniques to avoid detection and self protection of their code. This section has discussed many of these techniques. When extracted from the viruses and understood, these techniques have potential to be used for legitimate purposes in critical software application protection mechanisms.

2.5.2 Artificial Immune Systems in Computer Virus Detection. Artificial Immune System (AIS) design encompasses self-organization, self-regulation, primary and secondary immune system responses, clonal selection, adaptation and diversification,

knowledge extraction and generalization, a network structure, metadynamics, memory, and knowledge of self and non-self [CaT02, Das99, PeP93, ShL05, TiK04] as evolved from biological immune system (BIS) constructs. Many of these processes are a hierarchical network integration of others and use a combination of the same operators to define their associate interactions. For example, *clonal selection*s followed by somatic hypermutation are elements of self-organization, adaptation and diversification, and positive selection and negative selection relate to self and non-self. Note that clonal selection followed by somatic hypermutation with high selection pressure is usually defined as *affinity maturation*. The antigen-antibody BIS pattern matching is accomplished by specialized proteins attached to B-cells which are white blood cells that can be stimulated to produce antibodies. These antibodies produced by the B-cells then bind with antigens. Such proteins are generated by Deoxyribonucleic Acid (DNA) through Ribonucleic Acid (RNA) processing (Figure 2.16). These proteins then, as appropriate, become gene libraries for use as initial antibodies. B-cell computational antibodies usually correspond to an AIS analyzing computer input streams for viruses, where as file structures with embedded viruses map to the combined B-cell and T-cell helper (white blood cells that determine which antibodies the B-cells produce) situation matching the Major Histocompatibility Complex (MHC) protein, the equivalent partial antigen [Das99, TiK04]. The major vehicles for the AIS adaptation process to virus protection relate to changes in the computational antibody structures through some form of mutation. In particular, the use of an affinity maturation model of antibody clones provides new virus exploration. This brief mapping of AIS to BIS discussion is quite

limited with more details found in [Das99, TiK04]. In the proposed augmented virus protection AIS of Chapter IV, the reverse transcription RNA process is formulated as an innovative adaptive computational operator.

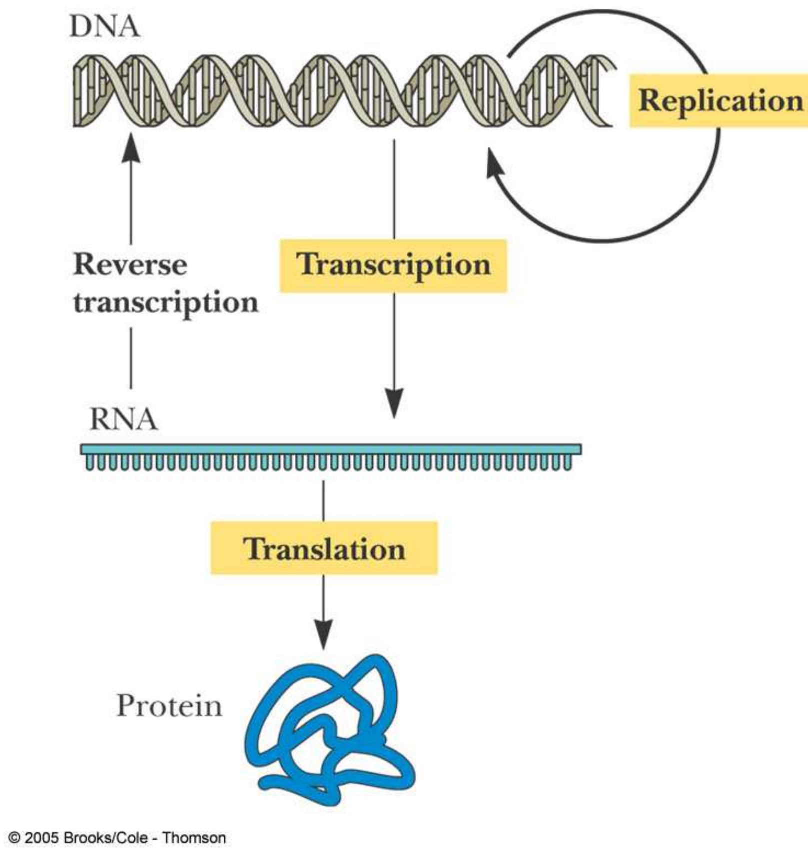

Figure 2.16: Reverse Transcription in the Human Body [UnS05].

Most contemporary virus detectors use a scanning engine to match input streams or file structures against known virus signatures (antigens) [Szo05]. A major AIS computational problem with this approach is every virus signature must be stored and compared to every computer input stream or file. This implies a lot of pattern matching computation and yet it still fails to detect viruses that have not already been identified and cataloged.

This brief background of an AIS and how it is used in computer virus detection will be used in Chapter IV in discussing the development of the biologically inspired virus scanner.

2.5.3 Mobile Ad Hoc Networks (MANETS) Background MANETs are a network of individual mobile wireless nodes that communicate with each other without any inherent network infrastructure or centralized control [DeL02]. There are many applications for MANETs as they allow the exchange of information real-time in a mobile environment. A hypothetical military application is depicted in Figure 2.17. In this application, multiple entities such as unmanned aerial vehicles (UAVs), robots, and even humans on the battlefield exchange the real time information they need in a dynamic environment to operate more effectively.

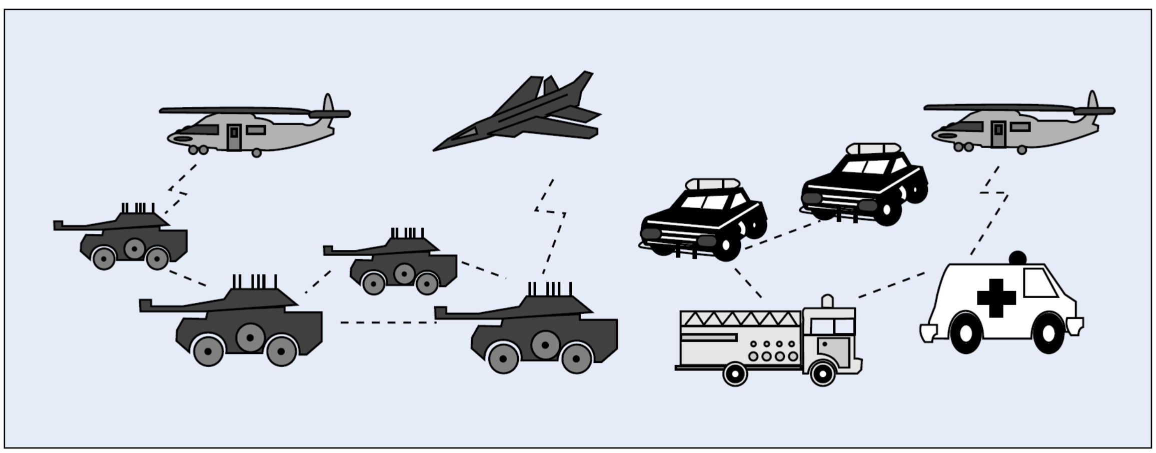

Figure 2.17: Example MANET [DeL02].

The major advantages of a MANET are unrestricted mobility and connectivity [DeL02]. The most significant disadvantage of a MANET is its more complex security issues due to changing network topology, limited capability of individual nodes, and its reliance on a trust relationship between nodes. These additional security issues are

discussed in the following paragraphs.

MANETs are vulnerable to attacks based on their fundamental characteristics of constant changing topology, lack of centralized control, required distributed cooperation, limited individual node capability, and open medium [DeL02]. The changing topology presents challenges in routing as well as implementing any type of static security solution. Without centralized control, individual nodes must rely on other nodes in the network to communicate. If a malicious node fails to follow the correct protocols, it can wreak havoc to the system if the other nodes do not recognize the suspect node as malicious. The limited capability of nodes opens up the possibility of a denial of service attack by exhausting limited resources such as battery life. It also makes detection of a malicious node harder because nodes often disconnect from the network to conserve their resources during normal operation, not just because they are behaving badly [ZaL03]. Finally, the open medium of the network allows anyone to listen to communications and possibly join the network as an imposter. Many of the types of vulnerabilities that a wireless network is susceptible to are the same as those for a wired network. They include eavesdropping, spoofing, replay attacks, and denial of service [AlC02]. Because mobile network routing relies heavily on a trust relationship between nodes, mobile networks are especially susceptible to routing misbehavior as communication can be disrupted or even impossible when it occurs [SaB05]. Routing behavior may be the result of a malicious node that is corrupted via an intrusion or a simple system failure. For the purposes of this research it is assumed that the misbehavior is due to an intrusion.

Before trying to design a system to detect intrusion on a mobile network it is

important to define an attack. An attack is defined as ‘a violation of expectations of the agent programmer or owner caused by one or more than one intentional attacker(s)’ [MaW01]. It is important to note that this definition only refers to ‘intentional’ attacks. Thus an anomaly that is caused by a normal system failure is not covered in the scope of this research. Due to the nature of MANETs, some attacks will be successful. Because MANETs are vulnerable, an intrusion detection system is vital so operating nodes can ignore nodes that are malicious or have been compromised [DaM05].

Due to the distributed nature of a MANET and the requirement to protect every node individually, an artificial immune system approach to intrusion detection for the network is a natural approach to the problem. The use of an AIS for an intrusion detection system (IDS) allows the system to learn what normal behavior for the system is based on past patterns of activity and detect anomalous behavior from a malicious node much as the human immune system learns what types of cells are allowed and detects malicious cells that are trying to attack the body. Due to the lack of centralized control in a MANET, the immune system agent (ISA) must be host based as shown in Figure 2.18.

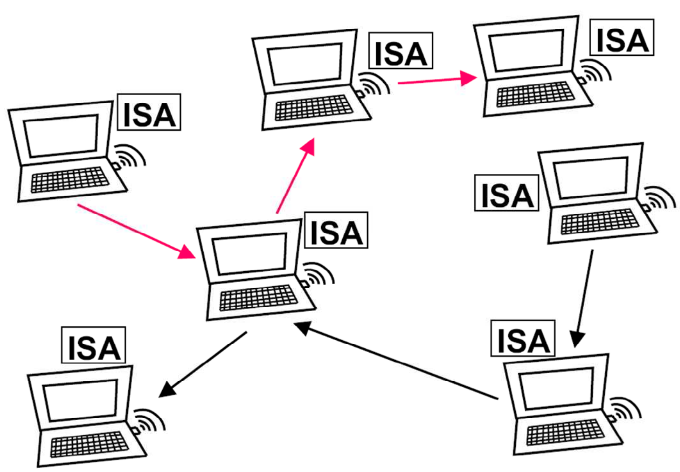

Figure 2.18: Host Based Agents in Mobile IDS [SaB03].

2.5.3.1 Critique of Current Literature. As mobile networks continue to grow in importance, the number and types of attacks also continue to grow. Intrusion detection for mobile networks has become an issue that also continues to grow and the amount of scholarly literature reflects it. This section critiques selected articles that represent some of the most advanced concepts in intrusion detection on MANETs.

The closest representation of using a multi-objective artificial immune system (MOAIS) to solve the problem of mobile network intrusion problems is found in [SaB03]. The authors continued their research in [SaB05] and improved the AIS described in this research to include a virtual thymus to eliminate the need for the protected learning phase, added a danger signal to decrease false positives, used memory detectors to decrease the time until detection of malicious nodes, and added clustering to further reduce false positives. The journal article expands their work from [SaB03] and presents promising results. As an appendix, they include pseudo code of their AIS building blocks.

In [KaG03], Karchirski and Guha propose a system that uses an agent type of intrusion detection system. Clusters of nodes use a protocol to select which nodes act as the agents and what their respective functions are. Although an interesting concept, there are some serious shortcomings in their work. First, their results indicate the system is not very scalable. After adding about 40 nodes, the number of network communication packets that are dropped becomes unacceptable as shown in Figure 2.19. This makes the system an easy target for a coordinated denial of service attack using two agents. The first agent would flood the specific IDS agents with packets while the second would

actually perform a specific attack which would go undetected with a high probability.

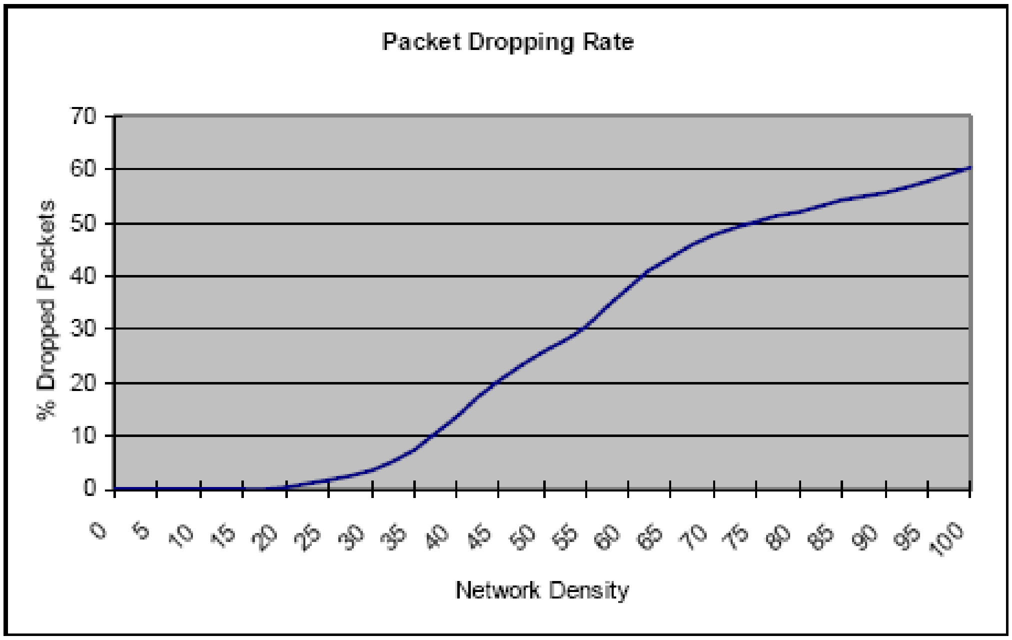

Figure 2.19: Non-scalability of IDS [KaG03].

The authors also state that if any node is suspicious, the system will reissue security keys for the entire network. That is a tremendous amount of overhead for a large network with multiple keys for simply a suspicious node. Again, this shows the proposed IDS system is not very scalable.

Another shortcoming of [KaG03] is that the issue of how to deal with a malicious node during their voting and selection protocols is not addressed. Every node is not an IDS agent so every node does not monitor the network. Depending on the mobility of the nodes and where the agents are, a malicious node can remain undetected indefinitely. If the malicious nodes can influence the selection of the agents through rigged voting then they have an even greater chance of remaining undetected.

In [MaP02], the authors present a sound design using a statistical anomaly

approach to a network IDS. A shortcoming of older IDSs is they typically assume a normal distribution for events that may not actually follow that distribution. The proposed design uses neural nets and the Kolmogrov-Smirnov (K-S) test and so are more based on the cumulative distribution function for their metrics which are much easier to develop empirically. This research is applicable to the design presented herein since the method from [MaP02] could be used with the MOAIS to develop gene patterns for detection. Currently, these gene patterns are defined using a best guess methodology to attain the best detection capabilities [SaB03]. The system could be improved if some sort of dynamic statistical analysis were used to determine the genes.

A good background paper on the issue of IDS in MANETs is [ZaL03]. The unique vulnerabilities of MANETs are described thoroughly and appropriate architectures for an IDS are identified. The authors conclude that every node needs to have some form of IDS. This is a direct contrast to [KaG03].

To keep their research manageable, the authors restrict the types of attacks used in their experiments to those against routing protocols. They also use only three types of protocols in their research: Dynamic Source Routing (DSR), Ad-hoc On-Demand Distance Vector Routing (AODV), and Destination-Sequenced Distance Vector Routing (DSDV) [ZaL03]. Based on these protocols, they determine it is important for the routing protocol to have some degree of redundancy for anomaly detection to work best [ZaL03].

Patwardhan, et al., propose an IDS for implementation on handheld computing devices in a MANET [PaP05]. To their knowledge, theirs is the first implementation of

an IDS for handheld devices and uses SecAODV. The basic operation of the IDS relies on comparing incoming packets to outgoing packets from a node's neighbors to determine if any neighbors are malicious. To do this, the handheld device listens in promiscuous mode to catch its neighbor's packets. For a handheld device, this seriously limits battery life -- an issue the authors fail to address. Although the authors argue that their solution is scalable, their experiments do not show this. They simply make the assertion but fail to substantiate it. Although it is important to develop an IDS for handheld type devices, the authors fail to show that their method is effective or viable.

2.6 Background Summary

This chapter summarizes the state of the art in attack trees as well as attack and protection metrics. Attack trees are still a relatively new research area. In the last 25 years there has been a moderate amount of development in the area but there is still much that is left to forge into new discoveries and developments. Metrics have had a fair amount of research done but few concrete results. Protection trees in system security are an entirely new concept as demonstrated by the lack of published papers. The SPM is a formal access control model that is used to demonstrate the effectiveness of the security methodology developed from this research. A basic background has been presented here to lay the foundation for the extension of SPM in Chapter V.

To initially develop the security methodology in Chapter III, additional areas of research were required beyond that listed in the above paragraph. Additional research was conducted in the areas of virus protections, human immune system inspired detection algorithms, and multi-objective systems such as intrusion detection in MANETs.

III. Framework Development

This chapter describes the security analysis framework developed through this research effort. First, the method of using attack trees in the security analysis is outlined followed by how metrics are used in these attack trees. Next, protection trees and the rule set for their metrics are explained. Finally, a generic security analysis on a system is conducted.

3.1 Attack and Protection Tree Methodology

3.1.1 Attack Trees. Once a high level threat is determined, it needs to be decomposed into intermediate objectives. These intermediate objectives can be further decomposed into individual attacker actions. Threat logic trees (TLT) are used to accomplish this decomposition [Wei91]. As explained in Chapter II, TLTs are a tree structure with the attacker's goal as the root node. Each child node is a decomposition of the parent node and are related to each other by either an *OR* or an *AND* relationship. In an *OR* relationship, when any of the child node tasks are accomplished then the parent node is successful. With an *AND* relationship, all of the child node tasks must be accomplished for the parent node to be successful.

TLTs today are known as attack trees [Sch99]. Although the name has changed, the structure and operation of attack trees are identical to TLTs. To illustrate how attack trees are constructed, a simple example is shown in Figure 3.1. The attacker's goal is to get a free television from an electronics store. The attacker can accomplish the goal by either stealing the television from a customer or by stealing it directly from the store. To steal the television from a customer, the thief must identify and follow a customer who

has purchased a television *AND* actually take the television from the person. This branch is an example of an *AND* relationship. To demonstrate the *OR* relationship, the thief has two ways to steal the television from the store. He can either take it from the shelf *OR* from the delivery truck. This example is simplified to clarify the construct of the attack trees. A real attack tree is more complete.

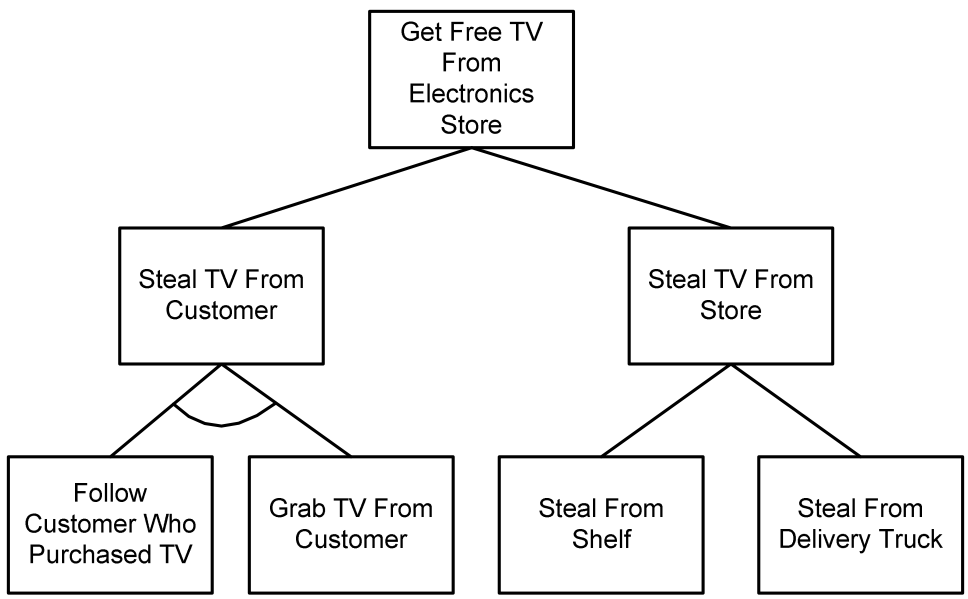

Figure 3.1: Example Attack Tree.

To better understand the construct of attack trees, a brief overview of the different level of nodes is necessary. While the root node is the attacker's ultimate goal, intermediate nodes represent sub-goals necessary to achieve the root goal. The leaf nodes are attacker actions. An attacker can only influence the system by interacting with it. These points of influence are represented exclusively by the leaf nodes. All nodes are derived from attacker actions, but an attacker cannot enter the attack tree at intermediate nodes.

3.1.1.1 Attack Tree Strengths. As shown in Chapter II, Attack Trees can be represented textually. It appears this format would lend itself easily to automatic processing, and can also be easily formatted automatically into Extensible Markup Language (XML) or some other portable language. The textual form of attack trees is also easier to create for more complex systems. Using the textual format, attack trees are easy to construct, they facilitate a methodical breakdown of threats, and readily allow security calculations and comparisons to be made. Numerical assessments based on cost, impact, severity of attack, and so on are achievable by assigning values to the nodes [Kar05]. Attack trees are an excellent tool for brainstorming and evaluating apparent threats, since they allow unconstrained assignment of threat components and values to the threat components. Since assigned values can be numeric or textual, they permit both technical and non-technical analysis. This allows a wide variety of attacks to be enumerated and analyzed, and the weakest link determined [Kar05]. Attack trees also allow the playing of "what-if" games with potential countermeasures [Sch00]. Additionally, the hierarchical structure is easy to navigate and allows multiple experts to work on different branches in parallel [StS02]. Attack trees allow the developer to refine the attacks to the level of detail desired, and exhibit the property of referential transparency [LiM01]. Referential transparency means that lower level details are abstracted away rather than omitted so that the higher level description contains everything needed to understand the entity in a larger context [PrT99]. This means developers can develop certain attack paths in greater detail while maintaining the context of the entire tree [LiM01]. According to [MoK01], attack trees can be used as a

checklist to help analysts detect attacks and train novice analysts. Standard attack trees can be deposited into a repository and reused which would increase productivity as the common attacks would not have to be recreated [MoK01].

Attack trees can also be automatically generated and analyzed for simple systems, ensuring they are exhaustive (cover all possible attacks) and succinct (contain only network states from which intruder can reach goal), as demonstrated by [ShH02]. First the network is modeled as a finite state machine (where state transitions = *atomic attacks* launched by intruder). An *atomic attack* is defined as a path in an attack graph that leads to an undesirable state. The attack graph is then produced and rendered. Finally, it is analyzed. Risk analysis, reliability analysis, or shortest path analysis are a few different types of analysis that can be performed to assess the vulnerabilities of the network.

3.1.1.2 Attack Tree Weaknesses. The tool is only as good as the analyst using it. Since attack trees are based on the domain knowledge of the analyst, any error or omission could result in a flawed tree and lead to an incorrect analysis. Additionally, a fully comprehensive attack tree with all possibilities and factors would require one or more experts and a significant amount of time and effort, resulting in poor scalability using the manual process [Kar05]. Another weakness is that there is no standard way of building, simulating, and analyzing these trees, so there is no simple way to share or reuse attack trees in a larger context.

3.1.1.3 Attack Tree Metrics. To properly analyze attack trees, metrics must be associated with each of the nodes in the tree. Metrics that are used include *probability of success*, *cost to attack*, *impact to the system*, and *risk*. The probability of success is a

number between zero and one and represents the attacker's probability of successfully completing the attack task at a specific node. The cost to attack is the attacker's cost to attempt to complete the task at a specific node. The units of cost may be dollars, man-hours, or a generic cost unit. Multiple cost units may also be combined within a tree by specifying a conversion factor such as the number of dollars per man-hour. The impact to the system defines how the system will be adversely affected if the attacker is successful at that node. It is a real number on the scale between one and ten, with one being the least impact and ten the greatest. This range is chosen because it is similar to the SWP impact function used in Weiss' work [Wei91]. The analyst subjectively uses definitions listed in Table 3.1 to arrive at the impact (I) number for the node. The impact definitions are intended to represent the total range of impact possible in any system. Risk is also annotated on each node and is calculated using the other metrics.

Table 3.1: System Impact Definitions and Numerical Ranges.

Numerical Range	Impact Definition
$1 \leq I < 4$	Minor impact to system. May be a nuisance but is easily detected and/or repaired
$4 \leq I < 7$	Moderate impact to system. Confidentiality, integrity, and/or availability of system affected. Requires non-trivial effort to detect and/or repair.
$7 \leq I < 10$	Severe impact to system. Significant damage results to system. Considerable effort required to detect and/or repair damage.
10	System completely compromised, inoperable, or destroyed

Depending on the system and the analysis desired, other metrics can be used as well. Such metrics may include *damage to the system* which defines the monetary loss as

a result of a successful attack node. An attacker may also be concerned with *probability of detection* which might deter an attack attempt at higher values.

Values for the chosen metrics are assigned at the leaf nodes because they are the only nodes that the attacker can actually control. It is only at the leaf nodes that the attacker interfaces with the system. The higher level nodes are intermediate goals that are a compilation of the attacker actions depicted by the leaf nodes. Once the leaf node metrics are determined, a set of rules is used to propagate the metrics up through the trees to the root nodes.

To assign metrics to the leaf nodes, they must first be measured in some way. Probability and cost are determined by using the best available method. Depending on the system, this may consist of either analyst estimation or historical data. The impact is assigned by analysts using a scale of one to ten from the general categories listed in Table 3.1. Analysts can determine the general range using the definitions and fine tune the numerical impact value depending on the actual case they are considering. Although general, these categories and definitions are subjective and may differ for other systems. Risk is calculated independently for each node using (3.1).

For the analyst to easily compare the risk values between nodes, the raw numerical value is normalized for the complete tree using a logarithmic scale as shown in (3.2). The lowest risk node in the tree is defined as $risk_{\min}$ and has a normalized risk value of one.

$$risk = (probability / cost) \times I \tag{3.1}$$

$$risk_{normalized} = \log(10 \times risk / risk_{\min}) \tag{3.2}$$

The metrics for intermediate and goal nodes are determined by the rule set shown in Table 3.2 that operates on lower level nodes beginning with the leaf nodes. After assigning values to the leaf nodes, the metrics are propagated up the tree until the goal node metrics are determined. An illustration of how the metrics propagate is shown in Figure 3.2. In each node, the raw risk value is shown along with the normalized risk value in parenthesis.

Table 3.2: Rule Set to Propagate Metrics up Attack Tree.

	AND	OR
Probability	$\prod_{i=1}^{n} prob_i$	$1-\prod_{i=1}^{n}(1-prob_i)$
Cost	$\sum_{i=1}^{n} cost_i$	$\frac{\sum_{i=1}^{n} prob_i \times cost_i}{\sum_{i=1}^{n} prob_i}$
Impact	$\frac{10^n - \prod_{i=1}^{n}(10-impact_i)}{10^{(n-1)}}$	$Max_{i=1}^{n} impact_i$

$prob \in (0,1], cost \in (0,\infty), impact \in [1,10], n = \#$ of child nodes

From this simple example, a few rules about the nature of the metrics can be observed. First, the parent of nodes with an *OR* relationship always has a non-decreasing probability of success compared to any of the child nodes. This is true because an attacker has multiple ways to be successful in an *OR* node. Therefore, his total probability of success will be equal to or higher than each individual child node. Conversely, the parent node of children with an *AND* relationship always has a non-increasing probability of success when compared with the highest probability child node.

Because every action must be successful for the parent node to succeed, the probabilities of the child nodes are multiplicative which lowers the total probability of the parent node as long as one of the child nodes has a probability lower than one. If all the child nodes have a probability equal to one (certainty), then the parent node will also have a probability of one.

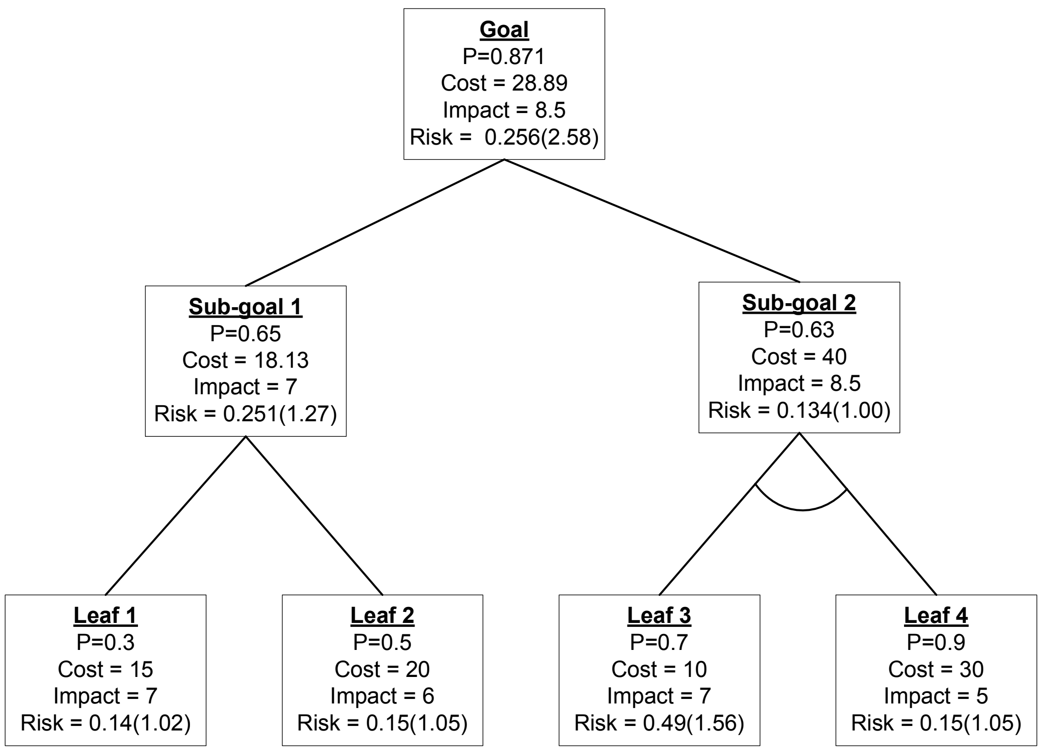

Figure 3.2: Example Tree Showing Propagation of Metrics in Attack Tree.

The particular attack path an attacker will attempt when he has a choice between actions (*OR* node) is unknown to the defender. As such, the costs for upper level nodes are based on a weighted average with the weighting derived from the probability of success for the attacker. This assumes that an attacker is more likely to attempt an attack that has a higher probability of success. In an *AND* node situation, the attacker must be

successful in every child node to succeed at the parent node level. This creates a situation where the costs are additive because every node must be completed.

3.1.2 Protection Trees. To best determine where to allocate resources for protection of a system, protection trees in conjunction with attack trees are introduced. The format of protection trees is similar to attack trees in that they are both *AND*/*OR* tree structures. Like attack trees, protection trees can be represented either graphically or textually. The differences between the two types of trees are in what the nodes represent.

A node in an attack tree represents a vulnerability. These vulnerabilities are specified but how to protect them is left out of the formal analysis. By specifying either a specific protection as a node or a sub-tree of protections, a protection tree analysis can yield where protections should be placed to get the greatest protection for the least expenditure of resources. The root node of a protection tree directly corresponds with the root node in an attack tree, but the rest of the tree's structure may differ widely.

To illustrate the concept of a protection tree, a partial attack tree (used to explore ways to break into a safe) is shown in Figure 3.3. For simplicity, the only metric used on this particular attack tree is the *cost to implement* each node. Because probability is not present in this example, the cost metric for an *OR* node is determined by taking the lowest cost of the child nodes. The associated protection tree developed to protect the safe against the attack is shown in Figure 3.4. Because the child nodes under the root node of the attack tree in Figure 3.3 are related by an *OR* construct, the attacker only has to find a single path to the root node to succeed. Therefore, the protection tree must mitigate or eliminate all attack paths at this level. This means that the child nodes under

the root node in the protection tree must be related by an *AND* construct.

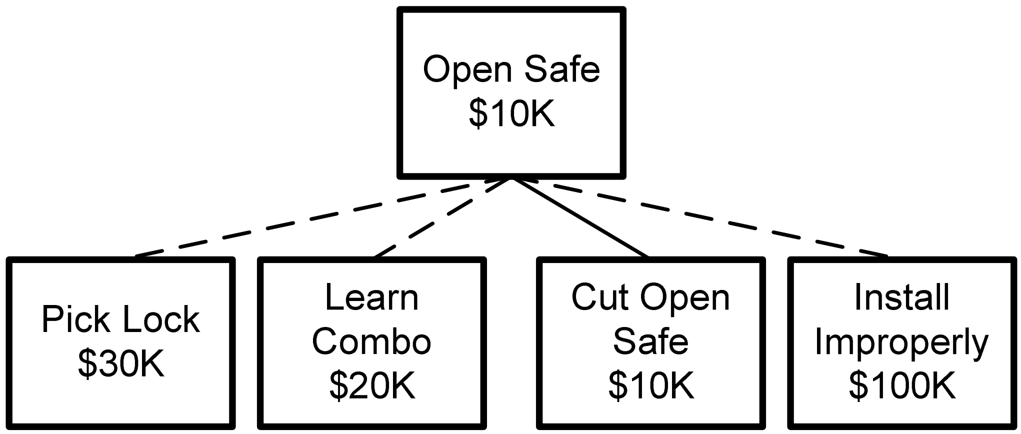

Figure 3.3: Partial Attack Tree to Open a Physical Safe [Sch00].

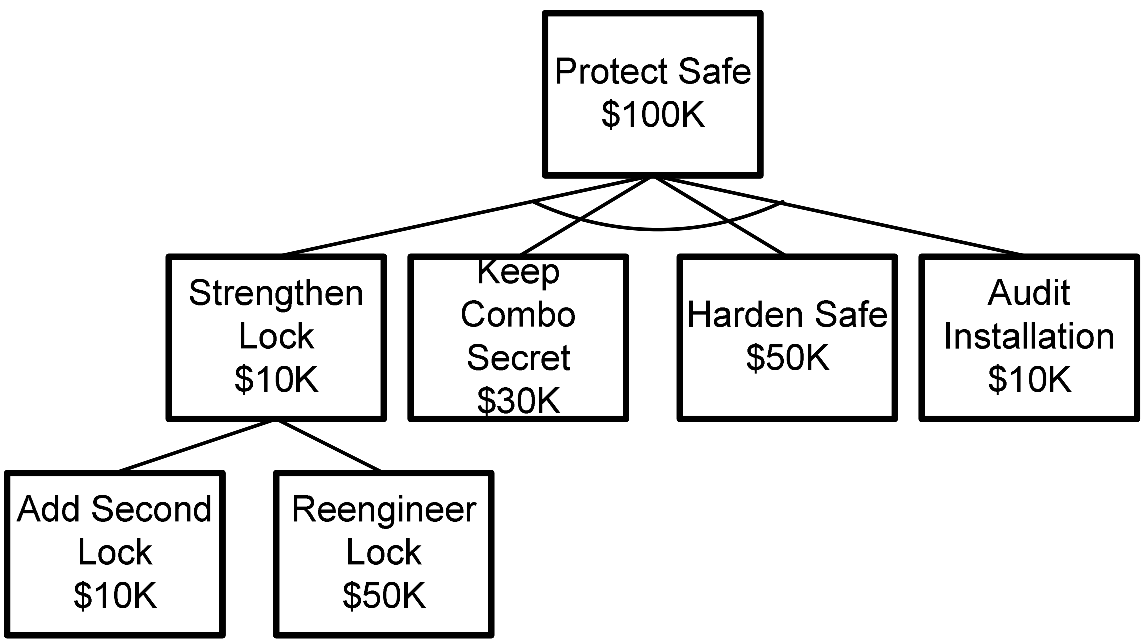

Figure 3.4: A Partial Protection Tree for the Safe Attack.

Although present in this example, there is not always a one-to-one correspondence between nodes in the attack tree and the protection tree. A single protection may be able to mitigate or stop more than one attack, or conversely it may take

multiple protections to mitigate or stop a single attack. There may also be options on which protection can be implemented to mitigate a particular attack. For example, the lock in the safe example can be strengthened by using a better lock or adding an additional lock. A cost analysis can determine which protection to implement. Protection trees facilitate this cost analysis as illustrated in the example.

To construct attack and protection trees, the basic algorithm is:

<u>Algorithm to build attack tree</u>

```
begin
        choose goal and set as root node
        ┌repeat
        │       decompose parent node into child nodes
        └until complete (no more child nodes can be created)
        assign metrics for all leaf nodes
        propagate metrics up tree to root node
end
```

<u>Algorithm to build protection tree</u>

```
begin
        assign protection to every leaf node in attack tree
        ┌repeat
        │       move up level in attack tree
        │       if protection child nodes do not cover parent attack node add
        │         protection nodes
        └until root node of attack tree is covered
        assign metrics for leaf nodes of protection tree
        propagate metrics up to root node of protection tree
end
```

3.1.2.1 Protection Tree Metrics. Two metrics are defined for use in the protection trees; *probability of success* and *cost.* Probability of success is the likelihood the protection will be successful in stopping the attacker at that particular node. Similar

to the attacker's probability of success, this number is between zero and one. Cost is the defender's cost to implement the stated protection and can be in a variety of units. There can also be multiple costs in a system such as financial costs and performance costs.

For the initial protection tree, the metrics are calculated as if every protection is implemented. To do this, the rule set from Table 3.3 is used. By implementing every protection, the maximum protection level and the maximum cost figures are determined. This establishes a baseline to compare future protection schemes against. The probability rules are the same as for attack trees. The financial costs for all types of nodes are additive as every protection is implemented in this initial protection tree. The performance cost equations accumulate the performance penalties of each node and result in a percentage of performance lost. For example, if two nodes have performance penalties of 30% and 40% respectively then the total performance penalty is 58%. This is because the first penalty is 30% and the second penalty is 40% of the remaining 70% of capability which is 28%. If two penalties are added together the result is a 58% penalty. The performance cost equations in Table 3.3 account for this behavior.

Table 3.3: Rule Set To Propagate Metrics Up Protection Tree.

	AND	OR
Probability	$\prod_{i=1}^{n} prob_i$	$1-\prod_{i=1}^{n}(1-prob_i)$
Financial Cost	$\sum_{i=1}^{n} cost_i$	$\sum_{i=1}^{n} cost_i$
Performance Cost	$1-\prod_{i=1}^{n}(1-cost_i)$	$1-\prod_{i=1}^{n}(1-cost_i)$

$prob \in (0,1]$, $financial\ cost \in (0,\infty)$ $performance\ cost \in (0,1]$, $n =$ # of child nodes

3.1.2.2 Cost Factor Equations. To facilitate the analysis phase, a *cost factor (cf)* for each node determines how much protection per resource unit that node provides. With two cost metrics in a multi-objective problem, three cost factors are computed using (3.3) where *fcf* is the *financial cost factor, pcf* is the *performance cost factor*, and *tcf* is the *total cost factor*. The *tcf* is weighted based on a sensitivity analysis with interaction from the *decision-maker (DM)* and is discussed later. This weighting factor determines which objective is favored when optimizing the protection scheme. Some protections will mitigate several different attacks while having a fixed cost. When this is the case, the cost factor is multiplied by the number of protections (number of parent nodes) it provides and increases its cost efficiency accordingly. A higher cost factor means the system provides more protection per unit of cost.

$$\begin{pmatrix} fcf = p \times (prob / cost_{financial}) \\ pcf = p \times (prob / cost_{performance}) \\ tcf = p \times prob / (cost_{financial} + (weight \times cost_{performance})) \\ \text{where } p = \text{number of parent nodes} \end{pmatrix} \tag{3.3}$$

3.1.3 Generic Attack and Protection Tree Security Analysis. To demonstrate how attack and protection trees are constructed and analyzed, a generic set of trees is described. The attack tree is created by decomposing the attacker's root goal into sub nodes until the level of fidelity is that of the actual attacker actions. The sub nodes and leaf nodes are related by either *AND* or *OR* relationships. After all the nodes are defined, metrics are placed on the leaf nodes and propagated up to the root node using

the rule set in Table 3.2. Typically, an attack tree is constructed from the root node down to the leaf nodes and the metrics are calculated from the leaf nodes up to the root node. A generic attack tree is shown in Figure 3.5.

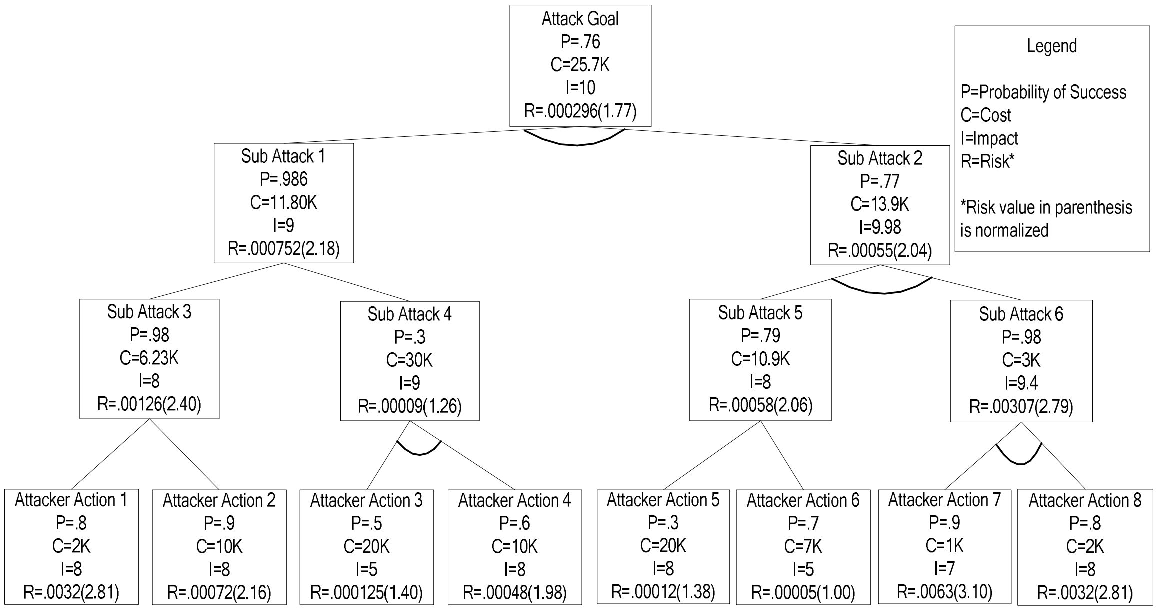

Figure 3.5: Generic Attack Tree.

To construct the protection tree, each leaf node in the attack tree is addressed with a possible protection and the next higher node in the attack tree is examined. If the higher node of the attack tree is covered by the existing protection nodes, the next higher level is analyzed until reaching the root node. If the attack is not mitigated, additional protections can be added with either an *AND* or an *OR* construct until the attack is mitigated. Once the attack tree is covered up to the root node using the protection tree, metrics are added to the leaf nodes of the protection tree and propagated up to the root node using the rule set in Table 3.3. During the first iteration, it is assumed every possible protection is implemented. This results in the maximum protection state

possible. The highest level of protection is available at the highest possible cost. A generic protection tree after the first iteration is shown in Figure 3.6. This generic tree is multi-objective with respect to cost and has to be optimized according to the DM's requirements. To calculate the *tcf* for each node, assume the weighting of \$5000 per one percent of performance is determined by cost analysis. A sensitivity analysis is presented later to determine the importance of this weighting factor.

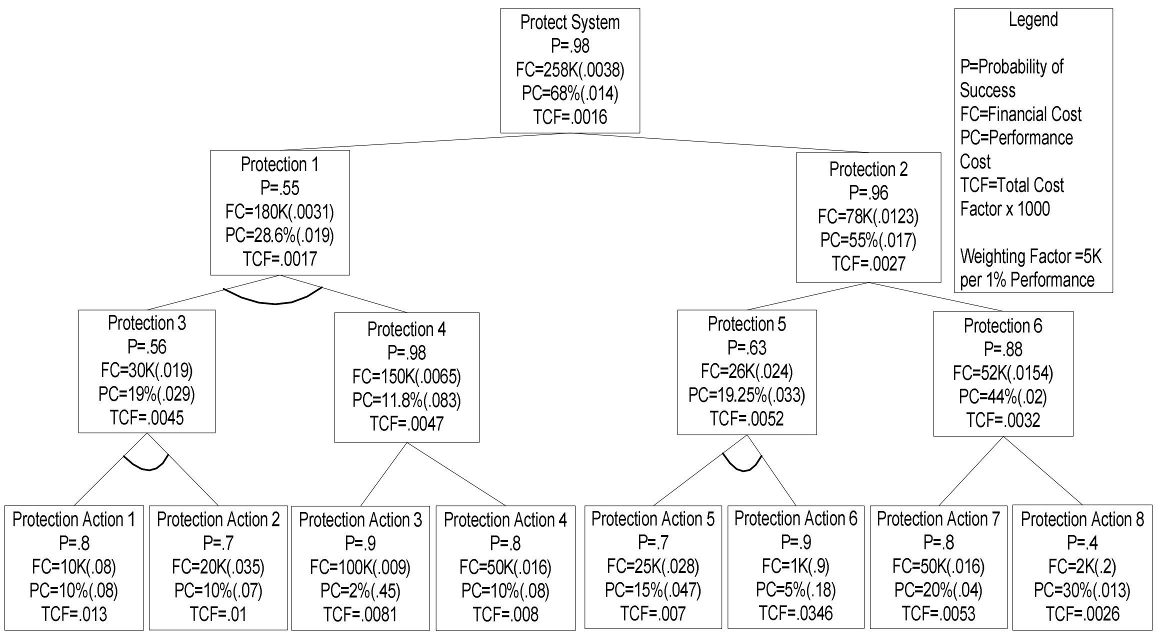

Figure 3.6: First Iteration of Generic Protection Tree.

Since protection resources are usually limited, the most efficient protections need to be selected when there is a choice such as in an *OR* node relationship. If the system is being optimized for either cost or efficiency, the respective cost factor is used to select which protections to implement. If both objectives are important, the total cost factor is used after determining a weighting for the cost factors. Assuming the weighting between cost factors remains the same, the optimized protection tree is shown in Figure 3.7.

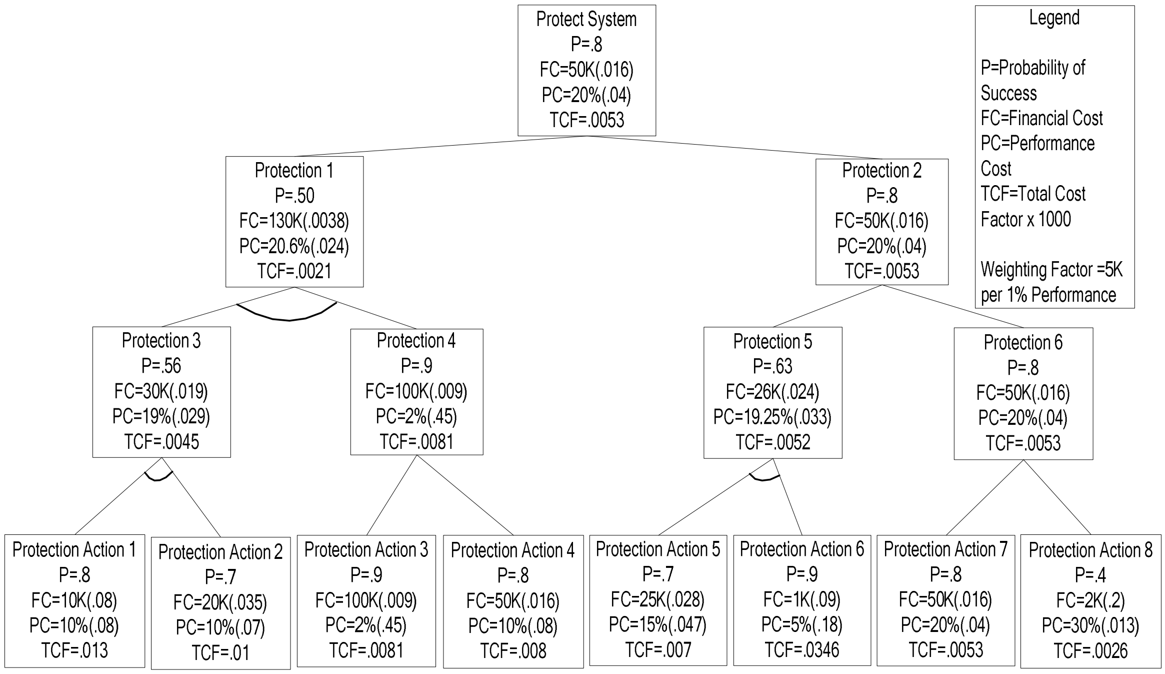

Figure 3.7: Second Iteration Of Generic Protection Tree Optimized for TCF.

A sensitivity analysis of the weighting factor must be accomplished to make an informed selection of the weighting factor. *Protection 2* dominates *Protection 1* for all values of the weighting factor due to its high financial cost and lower probability of success. The protection actions that make up *Protection 2* consist of a choice between three alternatives: *Protection Action 5* combined with *Protection Action 6*, *Protection Action 7*, or *Protection Action 8*. To find which protection to choose, the weighting factor is varied between $500 and $5000 per one percent of performance cost and the TCF for each alternative is charted in Figure 3.8. This chart clearly shows that *Protection Action 8* dominates until the weighting is greater than approximately $816 per one percent of performance. From $816 until $3820 per one percent of performance, the *AND* node comprising *Protection Actions 5 and 6* dominate. Above $3820, *Protection*

Action 7 dominates. Based on this analysis, a DM might be willing to change the weighting factor. For instance, if $4000 is the original weighting factor, the true financial cost of the protection scheme is $50,000 with a performance cost of 20%. If the DM accepts a weighting factor of $3800, the true financial cost is reduced to $26,000 with a performance cost of 19.25% by selecting different protections. Although this seems like a much better choice, the probability of successfully protecting the system drops from 80% to 63%. If the system can function at the lower protection level then the cost savings is significant for a small change in the decision-maker's weighting factor.

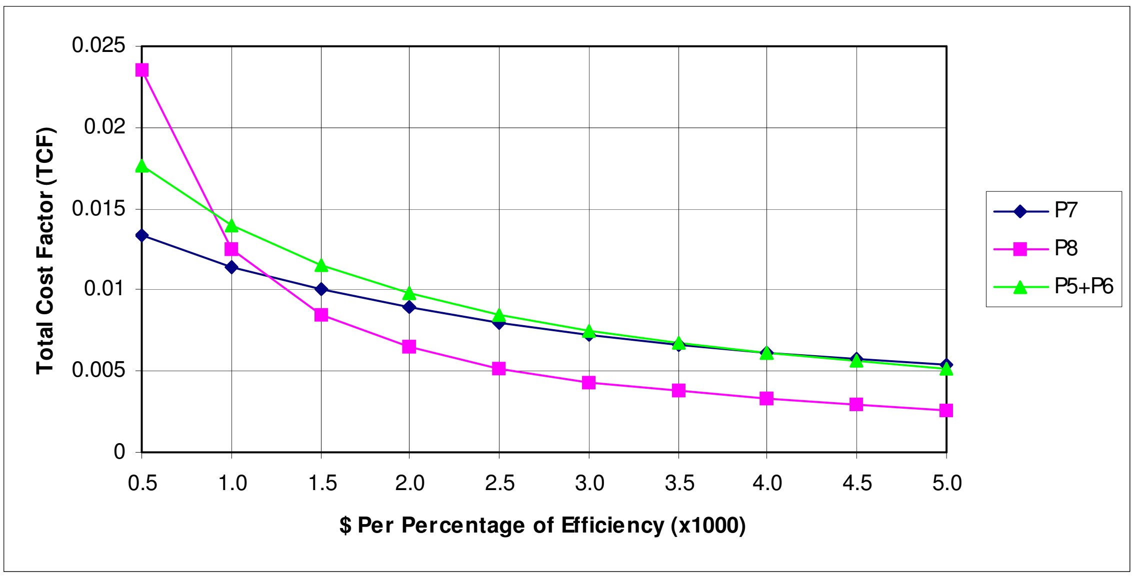

Figure 3.8: Total Cost Factor of Competing Protections for a Varying Weighting Factor.

3.2 Summary of Framework Development

This chapter explains the operation of attack and protection trees and how they are used in evaluating the security of a system. Specifically, the structure of the trees is explained, algorithms for the construction of the trees are given, metrics are defined, and

rule sets to propagate the metrics are given. To understand how a security analysis of a system would occur using this methodology, a generic set of trees is used in a multi-objective problem.

IV. Applications

This chapter details the use of the attack and protection tree methodology to analyze the security of four different applications. Using these four different application domains, the general usefulness of the methodology is demonstrated. Each application also denotes a milestone in the development of the methodology. As each application is used, different aspects of the framework are improved upon and further developed.

The first application is a general computer network attack. As this is the first use of the framework, the metrics are very simple and the analysis is limited in scope. The primary purpose of this analysis is to demonstrate the basic operation of the attack and protection tree methodology.

A Homeland Security Information Network is the next implementation of the methodology. This analysis is more detailed and a new *risk* calculation gives the defender an idea of where the system is the most vulnerable.

The third application involves an online banking network. In this iteration of the methodology, the *cost factor* is created. After working with many attack and protection trees in the previous development of the methodology it became apparent that comparing the efficiency of the protection nodes was difficult. To make this process easier and to ensure the most efficient protections are selected, the cost factor is created.

The final application is a mobile ad hoc network. This application domain is specifically chosen because of the limited computing power of the individual nodes. This limited computing power means that performance cost when implementing the

protections is very important in addition to the financial cost. This creates a multi-objective problem and the methodology has to be modified. The weighting factor and sensitivity analysis transform the multi-objective problem into a single objective problem for optimization based on the feedback received from the DM using the sensitivity analysis for the weighting factor.

Following the application discussion, research that actually led to the development of the framework is presented. Although presented after the application for clarity, the research occurred prior to the development of the methodology. This research was the impetus for the security framework methodology developed as a methodical process was needed to implement the protections. The actual research areas include using virus protections as software protections, developing a multi-objective intrusion detection system for a mobile ad hoc network, and developing an algorithm to be implemented in a virus scanner that could detect metamorphic viruses.

4.1 Network Security Application

Militaries have become increasing dependent on various types of networks in the execution of warfare. To increase the war-fighter's situational awareness and enable commanders to make timely decisions, Network Centric Warfare (NCW) has evolved into a complex system of multi-layered networks that communicate with each other. While enabling the war-fighter and commanders, these networks have been developed at such a rapid pace and have become so complex that security has sometimes been integrated as an afterthought. A systematic method for integrating security into military networks needs to be developed. This method can be used during development of the

networks as well as on networks that have already been fielded.

4.1.1 Example Protection Tree for a Network Security Application. To illustrate how attack trees and protection trees can be implemented, a simple analysis of a computer network is shown in Figure 4.1 and Figure 4.2. It is important to note that the metrics assigned in these trees are notional as the focus of the research is not determining the metrics but rather how they are used in the attack and protection tree methodology. This computer network could be one of many different types of networks used in NCW today.

It is assumed that it is part of the Non-secure Internet Protocol Router Net (NIPRNET). The partial attack tree in Figure 4.1 depicts the compromise of administrator access for a computer on the network. To keep the example generalized, what kind equipment is running on the network is not specified. Some nodes are not fully developed in an attempt to keep the tree easy to read in limited space. For an actual analysis, the specific network resources, equipment, and operating systems would need to be specified.

The first step of the analysis consists of building the attack tree. If a tree has previously been built for the particular network resource, it can be reused and examined for updates. Otherwise, the attack tree starts with the root node as the goal of the attacker. From this goal, child nodes are determined. It is important to make the tree as complete and unambiguous as possible to capture all facets of the attack. A method is discussed in [Amo94] includes pushing the arbitrary nodes as far down the tree as possible. Under the root nodes, the paths for the attack must consist of the complete search space. For this network security example, the child nodes of the root node include

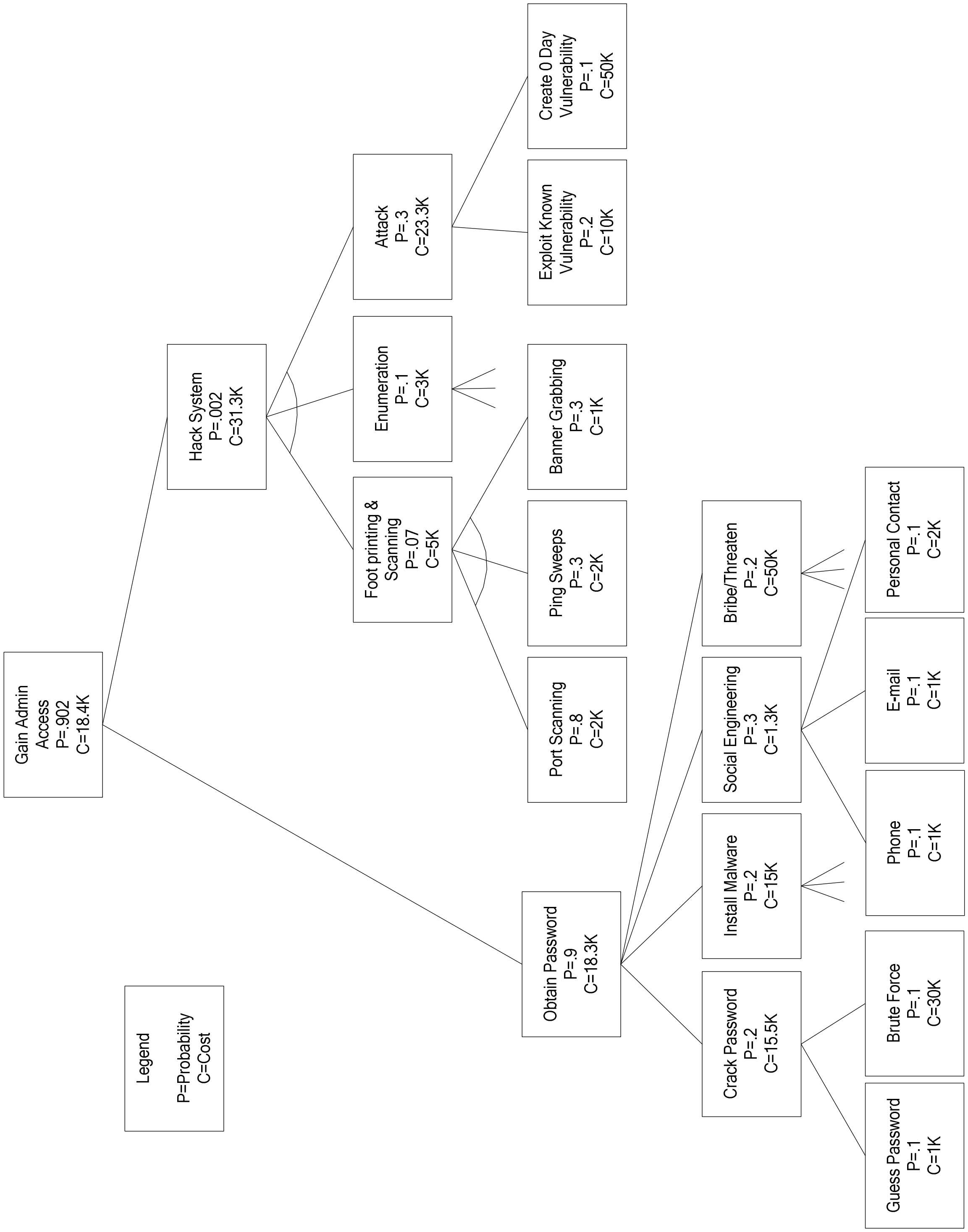

Figure 4.1: Partial Network Attack Tree.

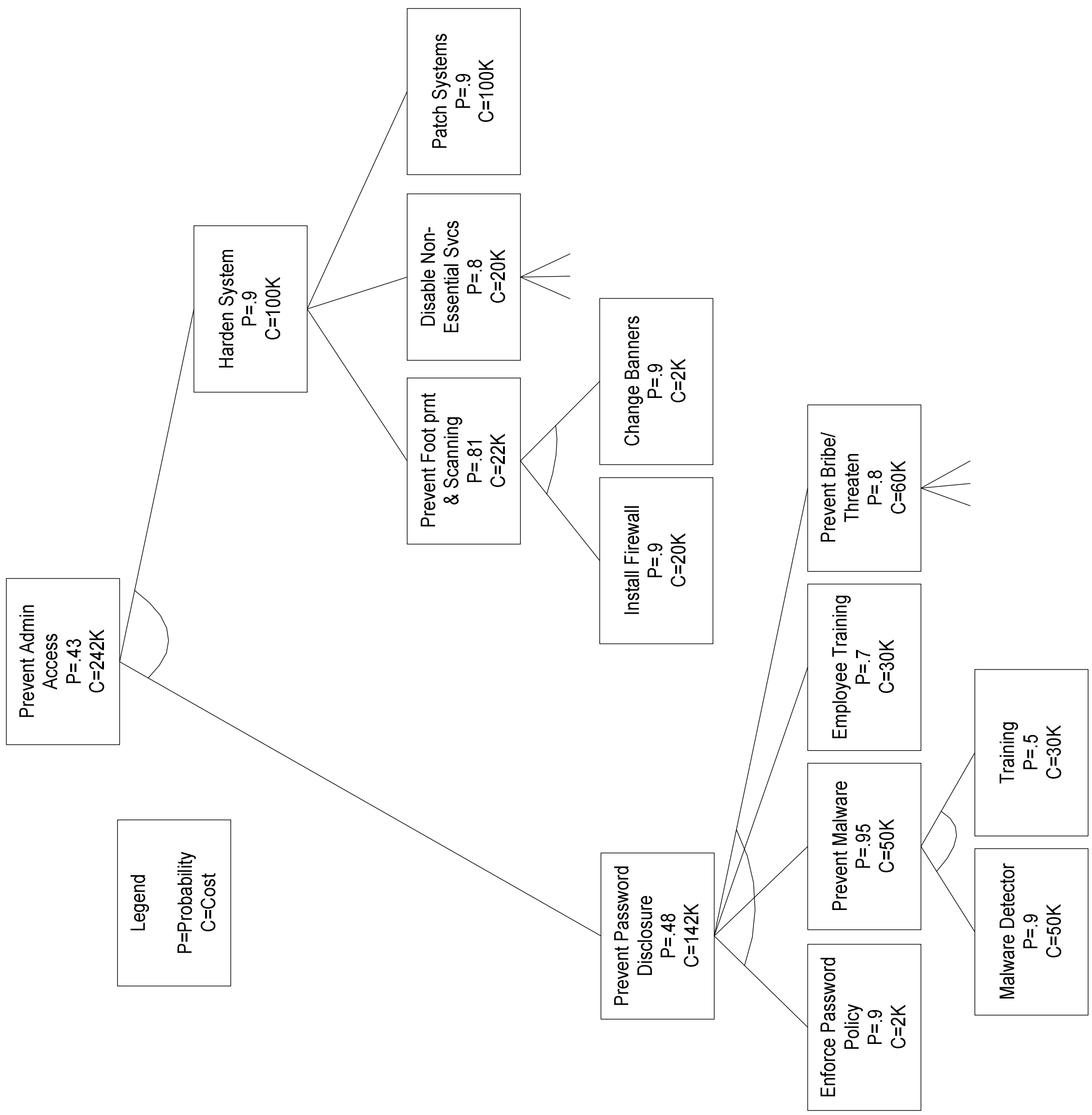

Figure 4.2: Partial Network Protection Tree.

hack system and *obtain password*. It is assumed that the two nodes represent the complete search space at this level. The relationship between the child nodes does not have to be binary to ensure completeness, but the union of the child nodes must cover every possible type of attack for the vulnerability listed in the parent node. Once the attack nodes are determined, metrics are attached to them. Attack metrics may include such things as cost to the attacker, probability of success, and probability of detection. To keep the example easy to understand only cost to the attacker is used. Other units of costs such as man-hours can be converted into units such as dollars to keep the analysis consistent.

After the attack tree is constructed and the associated metrics are assigned and calculated, the protection tree is built. The process is reversed from developing the attack tree in that the leaf nodes are addressed first. A protection, or sub-tree of protections, is placed next to each attack leaf node that will mitigate the attack. Once the leaf node is covered, the next level of the attack tree is addressed. If the combination of leaf nodes mitigates the higher level attack node then the next level up is addressed. If not, then a mitigating protection is added until the attack node is covered. This process continues up the tree until the root node of the attack tree is covered.

The next step is to assign metrics to the leaf protection nodes and propagate them up to the root node. Example metrics for protection trees may include *cost to implement* and *probability of success*. The method of propagation of the metrics will depend on the type of metric being used. For instance, if probability of success is used in the case of an *AND* node, the probabilities of the children are multiplied together and the result, the

probability that all children are successful, is assigned to the parent node. The cost to implement metric is determined by adding all of the costs of the children nodes. For an *OR* node, a cost analysis would have to be accomplished to determine which metric to propagate up. It is assumed the node with the highest probability of success will be chosen as long as the cost is under a specified threshold. The cost propagated up will be that of the protection node chosen. In Figure 4.2, *patch systems* is selected because it has the highest probability of success.

4.1.2 Analysis. From this example it can be seen that attacking is generally cheaper and has a higher probability of success than protecting the same system. This intuitively makes sense since an attacker has many different paths to enter the system and the defender has to protect all of them to completely stop the incursion. What is left out of this example is a metric for the probability of detection of the attacker. Depending on the attacker's motives, when the probability of detection is high, he may try a different route. The defender may not be able to stop the attack but can be fairly certain that the attack will be detected in a timely manner. This is the concept that most intrusion detection systems (IDS) use.

Notice that the probabilities of the child nodes do not necessarily sum to one in the attack and protection trees (Figure 4.1 and Figure 4.2) since the probability of an attack or protection being successful is independent of the success of any other attack or protection. Assigning probabilities to attacks and protections is problematic. An attacker will not always take a route that is well known. Although past events can be a predictor of attacker capabilities, skilled attackers need to be accounted for as well. A skilled

attacker will often attack components previously deemed secure or those that seem innocuous. For attacks that are yet unknown, leaf nodes that account for the uncertainty can be implemented. These leaf nodes can be updated as the attacks become known and the model is further refined.

By generating attack and protection trees, each of the decomposed components can be analyzed for vulnerabilities individually, and as a whole. Since these structures easily lend themselves to simulation using a myriad of current tools, "what-if" scenarios can be run to seek out potential vulnerabilities and develop appropriate protections before an attack actually occurs. This modeling is much like war gaming simulations that already occur. Running these types of simulations in a network recognizes that networks are a real part of the battle space in which conflicts are now being conducted.

4.1.3 Network Security Application Summary. In this particular example, attack and protection trees are implemented to analyze network security, specifically that of Network Centric Warfare. The protection trees shown here are simplified for ease of understanding. For more realistic uses, they must be extended. Improvements include methods to develop both attack and protection trees to ensure that they are unambiguous and complete.

Attack and protection libraries can be developed into many smaller components and placed in repositories. Large systems can thereafter be modeled from these repositories, which facilitates reuse-in-the-large and significantly reduces the time and cost of analyzing new systems.

4.2 Homeland Security

Due to their ever increasing importance in homeland security, computer networks are an important target to attackers. At the same time, constraints on budgets to protect these networks are very real. Therefore, a reliable, cost effective method must be used when deciding how to best protect the nation's critical computer networks from attackers. This application proposes using *protection trees* as such a method of determining the protections for computer networks used in homeland security.

4.2.1 Homeland Security Network Application. As an application example, using open source information only, partial attack and protection trees for the Department of Homeland Security's (DHS) Joint Regional Information Exchange System (JRIES) are created. This is the application suite that enables the DHS's Homeland Security Information Network (HSIN) to share information across Federal, State, and Local governments. All values used in the examples are notional and do not reflect actual metrics.

The primary mission of the JRIES network is the prevention of terrorist attacks [Men04]. Additionally, it can be used as a "collaboration, planning, and communications tool for facilitating the general homeland security mission across all jurisdictions nation-wide" and "serves as a means for supporting crisis management and recovery operations after a terrorist attack, as well as during and after a natural disaster" [Men04]. For this example, the attacker is assumed to have access to the HSIN, which is a protected network. JRIES uses Groove, which is a distributed collaborative software suite [Gri04, Gro06]. While it is harder to disable a system distributed over many clients, there are

still sources of data (databases on servers) that are vulnerable to attack. Individual or groups of clients can be attacked, producing a localized loss of data and capability. An attack on JRIES does not necessarily need to permanently disrupt the system, as it may be the goal of an attacker to only temporarily disrupt the DHS's response to an attack, or mask an attack in progress.

The partial attack tree in Figure 4.3 shows how an attacker might disrupt JRIES through a *distributed denial of service (DDoS) attack*, a *physical attack*, a *virus/worm*, or some *other attack*. Of course, *other attack* could be expanded in this case, but is not to keep the example simple. In practice, an analyst would make the decomposition of the tree as exhaustive as possible. The associated high level protection tree is shown in Figure 4.4.

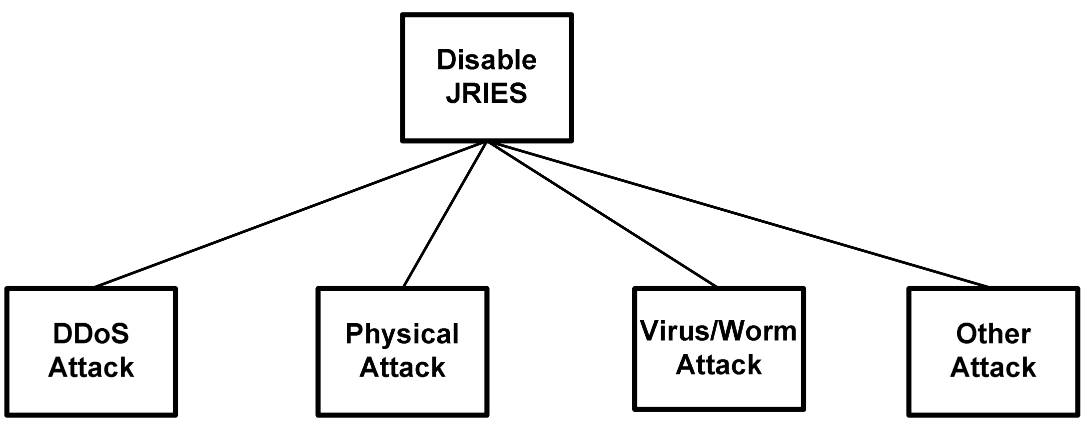

Figure 4.3: Disable JRIES Attack Tree (High Level).

To keep the example manageable, a single branch of the high level tree is developed. The DDoS attack in Figure 4.3 is decomposed further and shown in Figure 4.5. Although this branch of the attack tree is further developed, it is still not complete.

A full attack tree would implement sub-attacks that could be developed separately and maintained in libraries. An example of a node that would contain a sub-attack in a library is the *Attack Domain Name Service (DNS)* node.

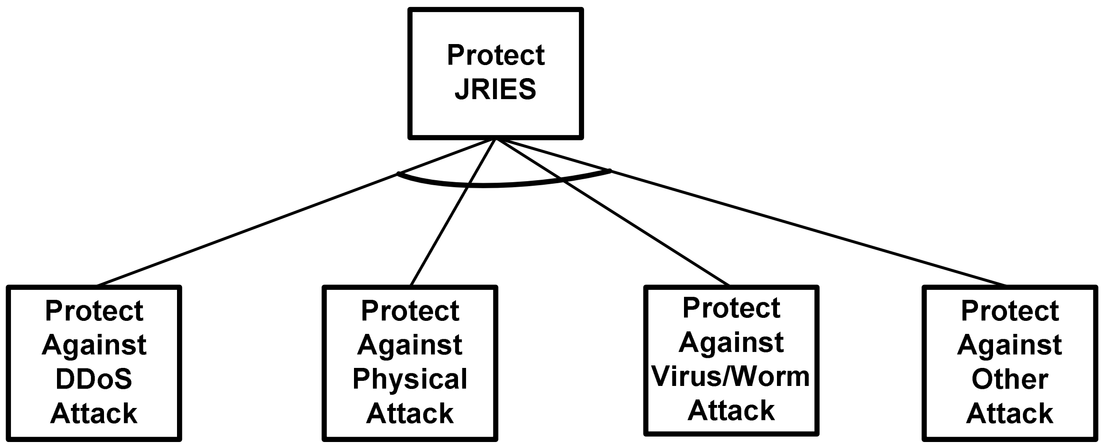

Figure 4.4: JRIES Protection Tree (High Level).

4.2.2 Results and Analysis for Homeland Security Application. Analyzing the DDoS attack tree in Figure 4.5, it becomes apparent that the greatest risk is an attack against the JRIES servers. Although the attacker's cost is moderate, the probability of success for the attacker is highest and the impact of the servers being attacked is severe. Going further down the attack tree on the server's branch, the highest risk sub category of attack is a DDoS attack using a BOTNET. A BOTNET attack is cheaper and has a higher probability of success than other attacks against the servers. Further analysis of this branch shows that the highest risk is for the attacker to "rent" an existing BOTNET rather than building his own. The impact of renting a BOTNET is lower than building one. This is because when renting, the attacker does not have complete control

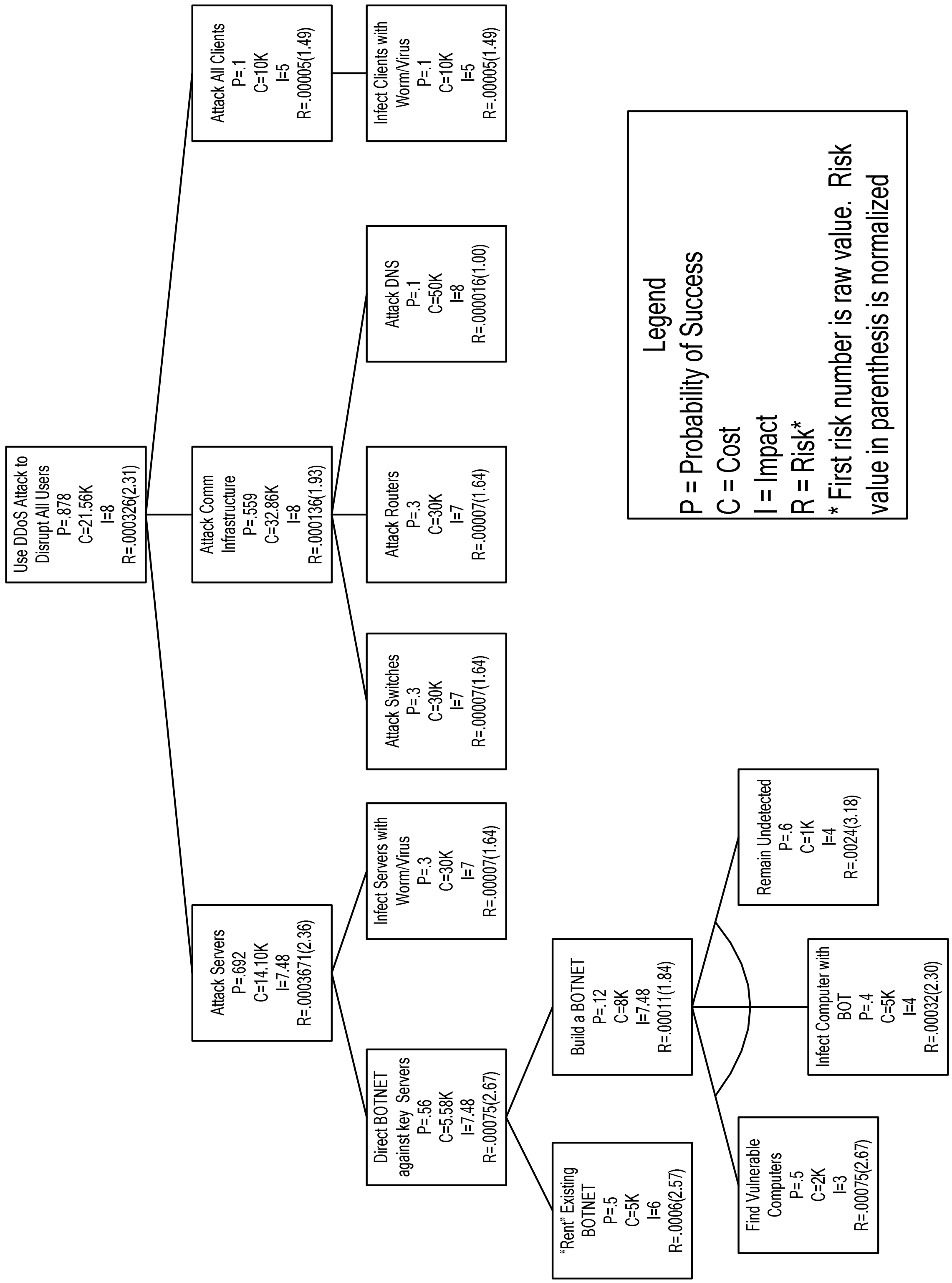

Figure 4.5: DDoS Attack Tree.

over the network. He may lose his ability to control the BOTNETS as well as possibly not having as much functionality as a custom BOTNET might provide.

When comparing the attack tree against the protection tree shown in Figure 4.6, it is apparent that protecting the system is harder and more expensive than attacking it. This is logical since an attacker only has to find a single way to attack the system while the defender has to defend against many ways into the system because of the uncertainty of how the attacker will attempt to cause the DDoS.

If resources needed to protect the JRIES system are unlimited, then every protection can be implemented and the system will have the greatest protection available. In this situation very little analysis is necessary. The actual case is usually quite different though. Resources are often limited and decisions have to be made on where to best use them in protecting the system. These situations are where the true value of protection trees can be realized.

To illustrate how a protection tree is used in this type of analysis, assume there are only $25K of resources available for protecting the JRIES system. The highest risk threat of the servers being attacked can be mitigated with the lowest cost by blocking the Internet Protocol (IP) addresses of the BOTNETS and inoculating the servers from worms and viruses. Using this solution, the servers can be protected with a 72% probability for $12K. The protection tree in Figure 4.7 shows this solution. After implementing this solution, the attack tree can be modified by either pruning the BOTNET branch or recalculating the probabilities of success for the attacker with the new protections in place. For simplicity, the branch is pruned and the attacker's

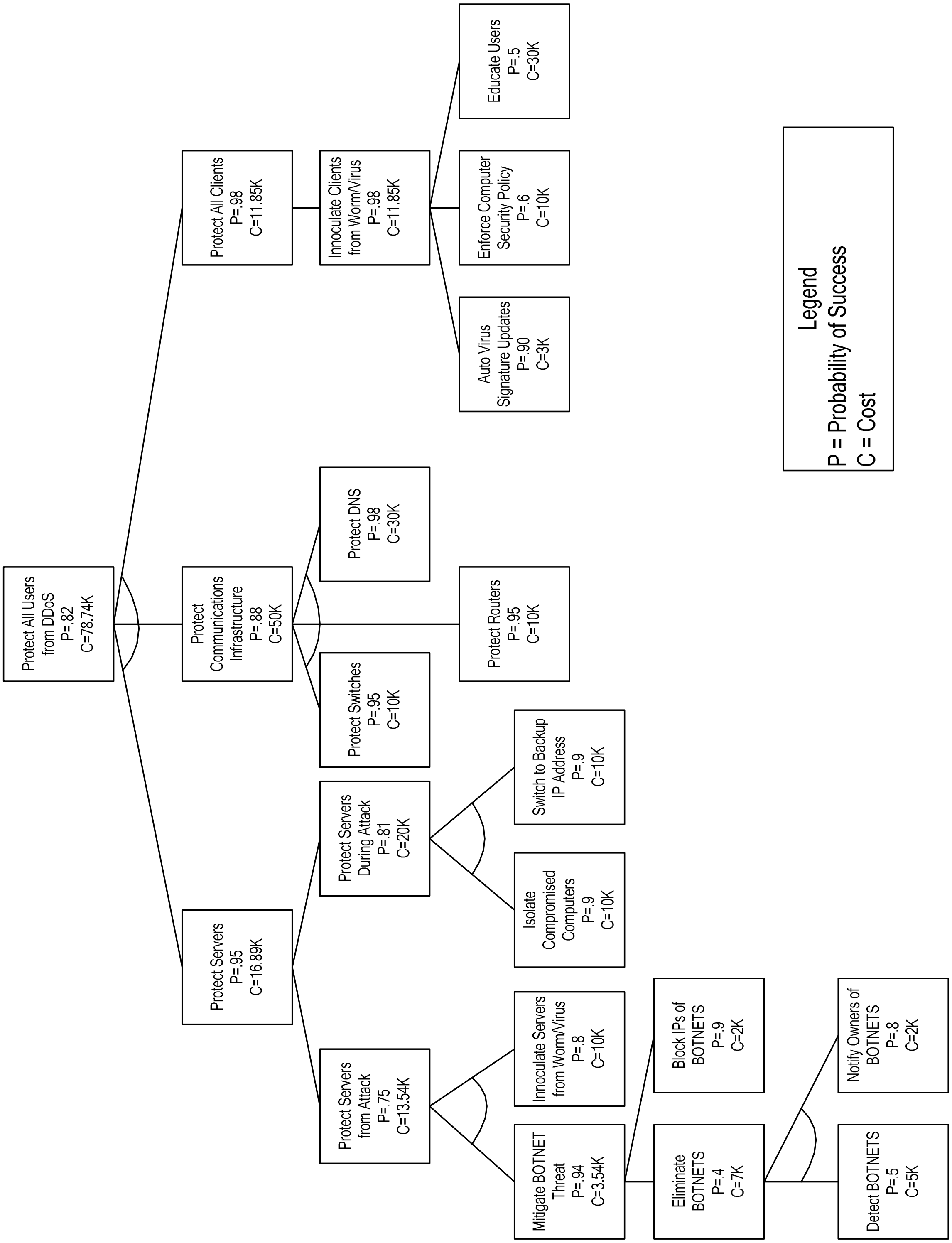

Figure 4.6: DDoS Protection Tree.

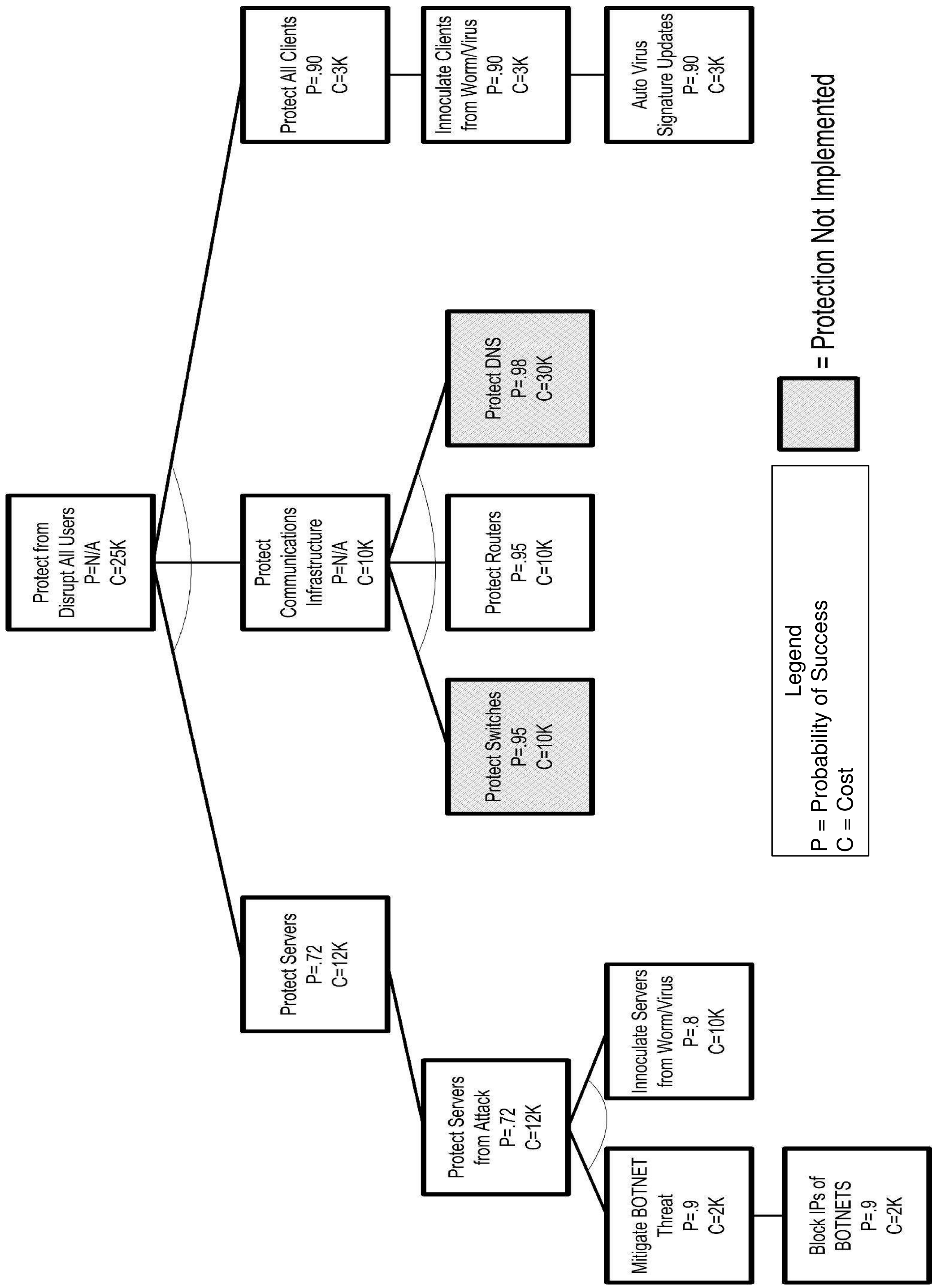

Figure 4.7: Pruned DDoS Protection Tree.

probability of success in attacking the servers is lowered to 30% (Figure 4.8).

The next highest threat is the attack of the communication infrastructure. Although not enough resources are available to fully protect the entire communication infrastructure, the attacker's probability of success can be lowered by protecting a single component. The switches and routers have the highest threats and the cost of protecting them is equal so the protection of the switches is chosen for a cost of $10K. As shown in Figure 4.8, this lowers the attacker's probability of success in attacking the communication infrastructure to 37%.

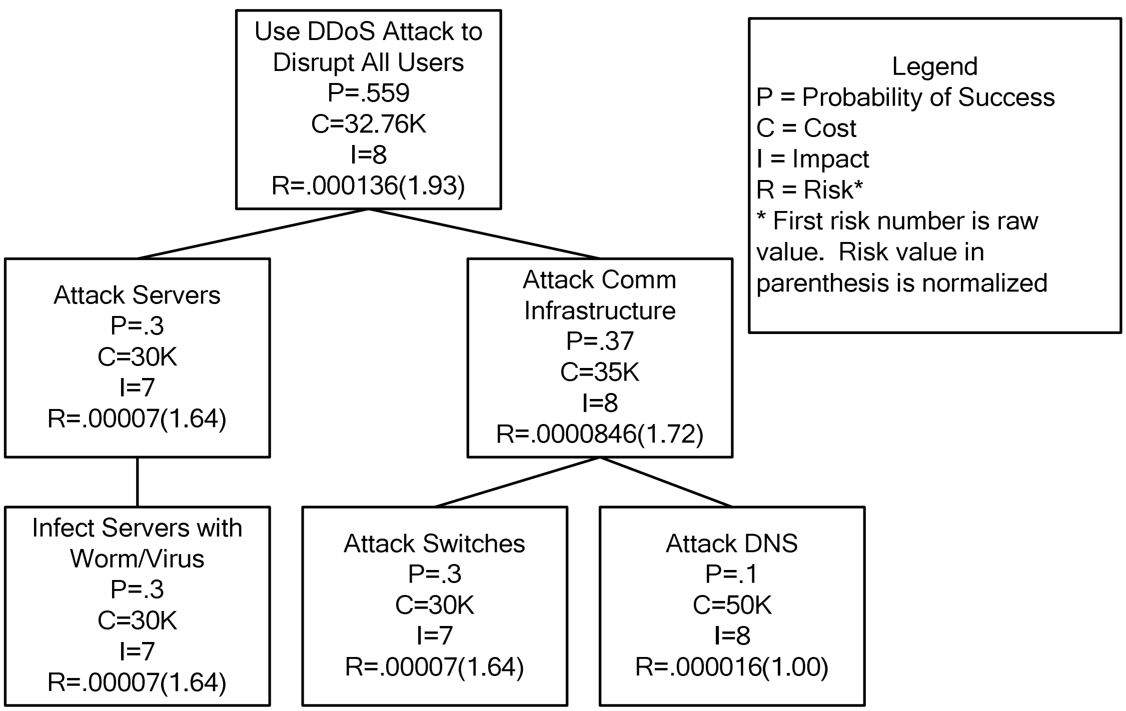

Figure 4.8: Pruned DDoS Attack Tree.

With $3K left to protect the system, auto virus signature updates can be implemented and the threat to the clients eliminated completely with a 90% probability. Analyzing Figure 4.8, the attacker's probability of success is lowered and his expected

cost to attack is increased when compared to the initial attack tree in Figure 4.5. This in turn has lowered the risk to the system.

Another method that can be used when analyzing which protections to implement is *pre-pruning* of the attack tree. This method makes assumptions about the attacker's capabilities. For example, it might be assumed the attacker will not spend more than $25K on any attack. If this is the case, then all nodes that cost the attacker more than $25K are eliminated. With this reduced attack tree, the nodes can be recalculated and same methodology used before can be applied to implement the appropriate protection measures.

4.2.3 Summary of Homeland Security Application. Using attack and protection trees in conjunction with the metrics, the DHS's JRIES system is analyzed as an example to seek out vulnerabilities and appropriate protections. This example analysis of the JRIES system demonstrates how attack trees and protection trees are used in the analysis of an arbitrary system.

4.3 Online Banking

Online banking has become increasingly important to the profitability of financial institutions as well as adding convenience for their customers. As the number of customers using online banking increases, online banking systems are becoming more desirable targets for criminals to attack. To maintain their customers' trust and confidence in the security of their online bank accounts, financial institutions must identify how attackers compromise accounts and develop methods to protect them. Attack trees and protection trees are a cost effective way to do this. Attack trees highlight

the weaknesses in a system and protection trees provide a methodical means of mitigating these weaknesses. In this section, a notional online banking system is analyzed and protection solutions are proposed for varying budgets.

4.3.1 Online Banking Application. As of 2004, an estimated 53 million Americans use online banking [Fox05]. This constitutes approximately one-quarter of all adults in the United States. The same survey also found that 55% of individuals with a household income over 75,000 United States Dollars (USD) use online banking [Fox05]. These individuals are an alluring target for criminals.

Being such a tempting target, online banking transactions must be protected to keep financial losses to a minimum and trust in the online banking system high. To efficiently do this, the threats to the online banking system and ways to mitigate the threats must be determined.

4.3.1.1. High Level Threats to Online Banking. The emergence of online banking is a key element in attracting and keeping customers for financial institutions. A recent survey shows the availability of online banking and online bill payment services is a more important factor in choosing a bank than actual physical branch locations [Key05]. Online banking greatly reduces transaction costs for the bank. A typical transaction at a physical branch is estimated to cost between one and four USD, while an online transaction costs less than 0.05 USD [Key05].

Clearly, banks want their customers to continue to use online banking. One of the biggest concerns for customers is their online banking sessions will remain secure. If customers do not have reasonable assurance their accounts are secure, they will not trust

the online banking system, leading to lost customers and higher costs for banks as customers revert to a traditional banking model.

Since banks want to attract new customers as well as retain current customers with their online banking services, they must provide reasonable assurance that their customers' accounts are secure. The main threat that concerns banks, with respect to online banking, is that an attacker will transfer money out of their customers' accounts.

4.3.1.2 Online Banking Attack Tree Development. Consider an attacker with a goal of illegally transferring money out of online bank accounts. Assume the attacker has no prior personal information on any of the victims before the attack, and the bank has approximately 50,000 online banking customers.

Since the attacker's goal is to transfer money out of customers' online accounts, this becomes the root node of the attack tree shown in Figure 4.9. It is assumed an analyst has developed the attack tree and assigned the appropriate metrics to all of the leaf nodes. As in the previous applications, the metrics assigned are notional. The metrics are propagated up the tree using the rules in Table 3.2. From the attack tree, the attacker has a 99.7% expected probability of successfully transferring money out of the accounts for an approximate cost of 63.17K USD. With an average theft of 1,200 USD per account [PeF04], the attacker would only need to compromise 54 accounts to pay for the attacks with near certainty. Obviously, this is not a favorable situation for banks. To counter the threats identified, a protection tree is developed.

4.3.1.3 Online Banking Protection Tree. To develop the protection tree, each node of the attack tree is addressed starting with the root node. If a node does not result

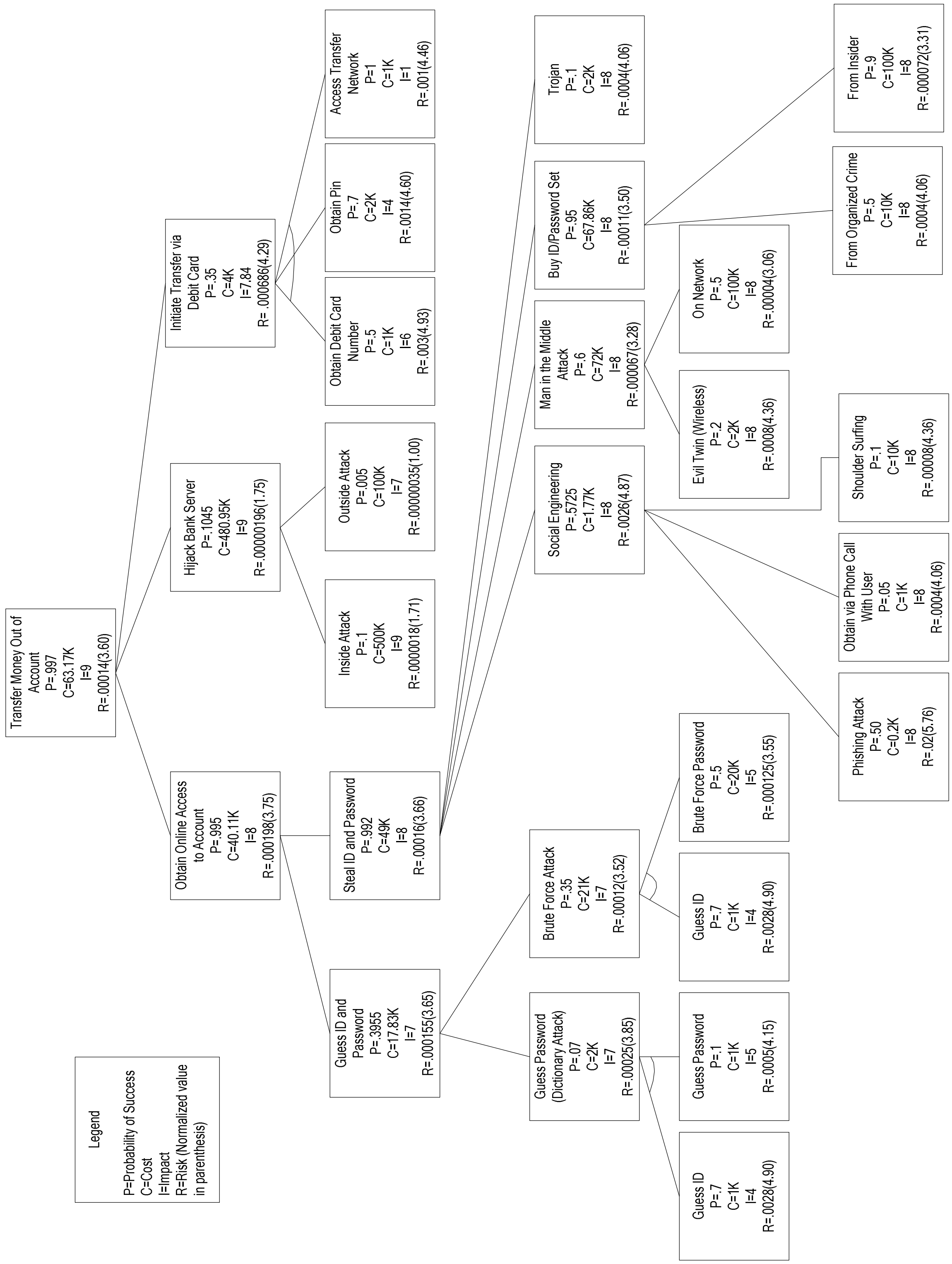

Figure 4.9: Online Banking Attack Tree.

in a specific protection, it is decomposed until it results in a specific defender action. Once this action is defined, metrics are assigned to the leaf nodes and propagated up to the root node. For the initial iteration, assume every available protection is implemented which results in the maximum protection state possible. From this, the maximum total cost for protection is determined and the highest probability of success is obtained. The initial protection tree for the online banking application is shown Figure 4.10. The cost for this protection scheme is 2.335 million USD for a probability of success of just over 50%.

Now that the initial protection tree is developed and the maximum level of protection is defined, the most cost efficient measures need to be determined. Propagating the leaf nodes with the highest cost factors at all *OR* nodes yields the most efficient protection scheme. This method ignores limitations such as requirements for a protection to be successful with a certain probability. For instance, there may be a requirement that states a protection node must have a probability of success over 80%. If that is the case, a specific protection may have to be implemented even if it is not the most cost efficient.

For *AND* nodes, all child node protections should be implemented. If resources do not allow all *AND* nodes to be implemented, the most cost effective ones are chosen and the attack tree with that protection in place is recalculated. The resulting attack has a higher cost and lower probability of success for the attacker. This shows that partial protection is possible for *AND* protection nodes and is somewhat effective in deterring the attacker.

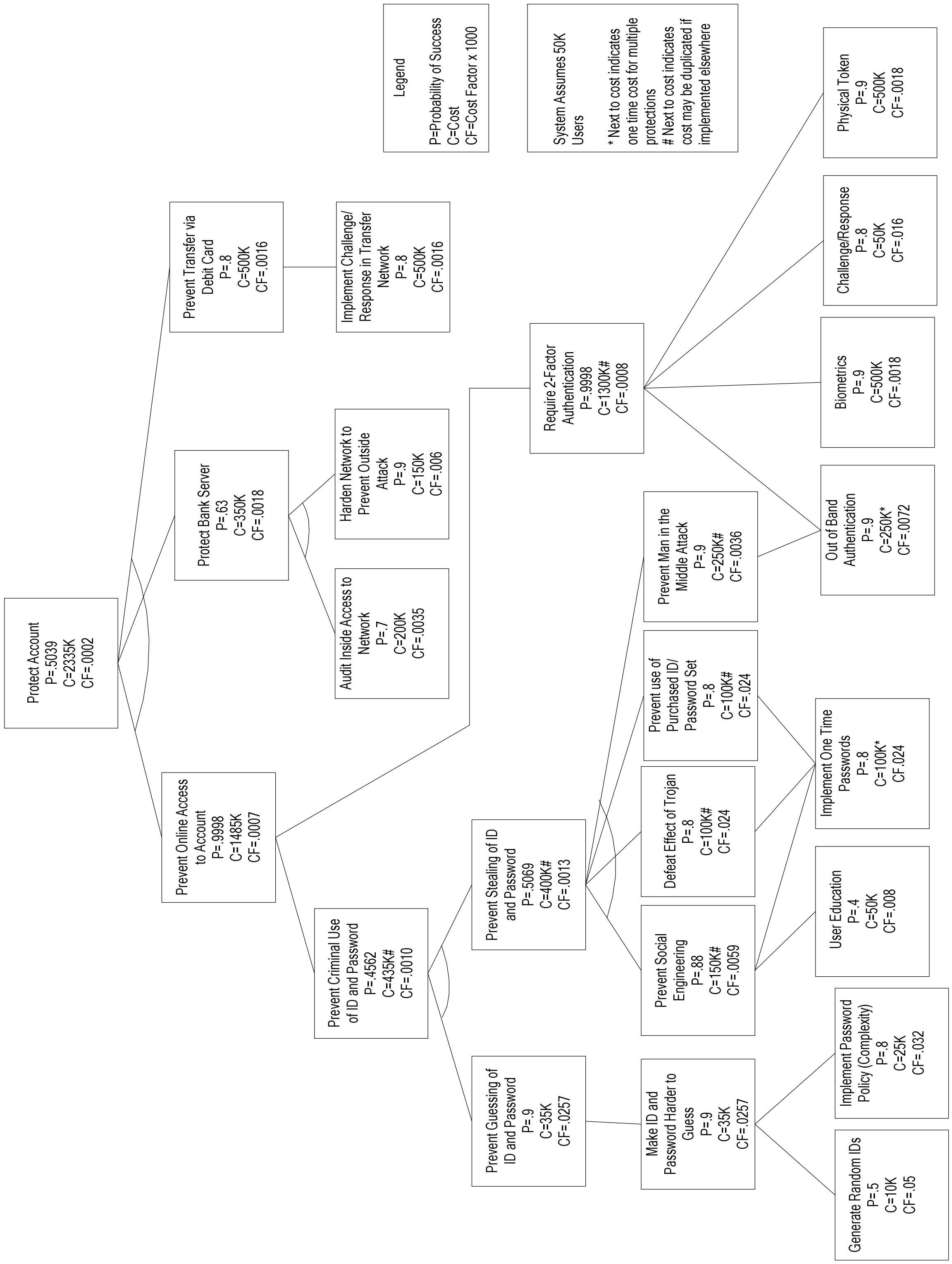

Figure 4.10: Initial Online Banking Protection Tree.

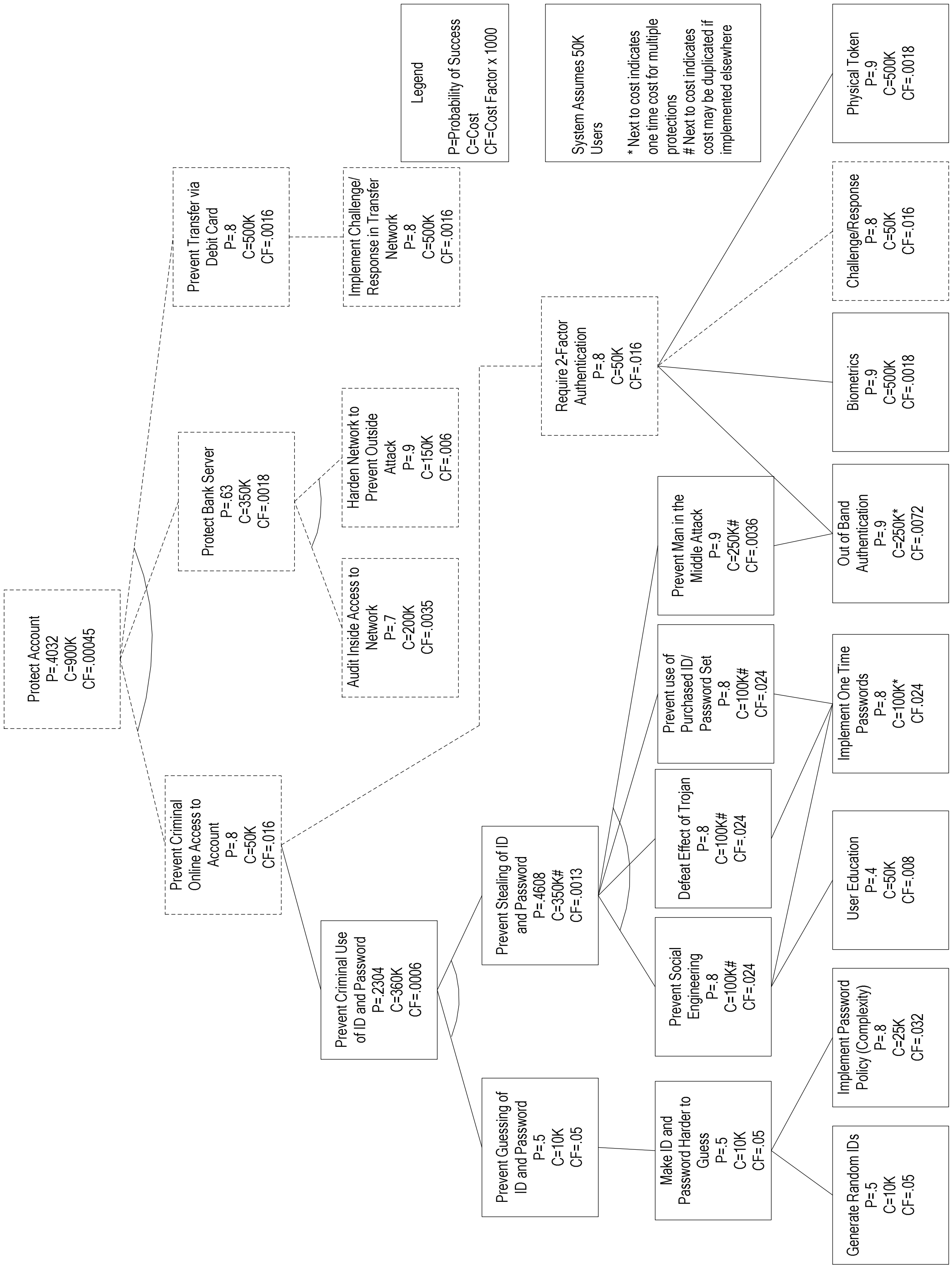

Figure 4.11: Online Banking Protection Tree after First Iteration.

After performing a cost analysis on the initial protection tree, the result is the protection tree shown in Figure 4.11. The protections implemented are represented with dashed lines. This tree indicates that the most cost effective solution to provide protection against all attack nodes for 900K USD which provides a probability of success slightly above 40%. The probability of success has dropped by about 20% from the maximum protection level but the cost has dropped over 61%.

If resources are further constrained, another iteration of the protection tree can be developed. For example, suppose only 200K USD is available. The resulting protection tree is shown in Figure 4.12. From this protection tree, complete protection against online access and partial protection against a bank server hijack is achieved. Because a probability of success cannot be obtained for this new protection tree, the attack tree is recalculated to measure the effectiveness of the new protections. This recalculated attack tree is shown in Figure 4.13. The attacker's probability of success has dropped over 60% and his cost to attack has increased by 180%. This confirms intuition; even a partial protection plan deters an attacker by reducing his chance of success and increasing his costs to attack thus making the attack less profitable.

4.3.1.4 Online Banking Analysis. The above scenario indicates the most effective defense is some type of multi-factor authentication. In fact, the Federal Financial Institutions Examination Council (FFIEC) states that the single factor authentication that most online banking systems use is inadequate [FFI06]. Although the FFIEC did not recommend specific technologies, it concluded that some type of multi-factor authentication is necessary in today's online banking environment to adequately

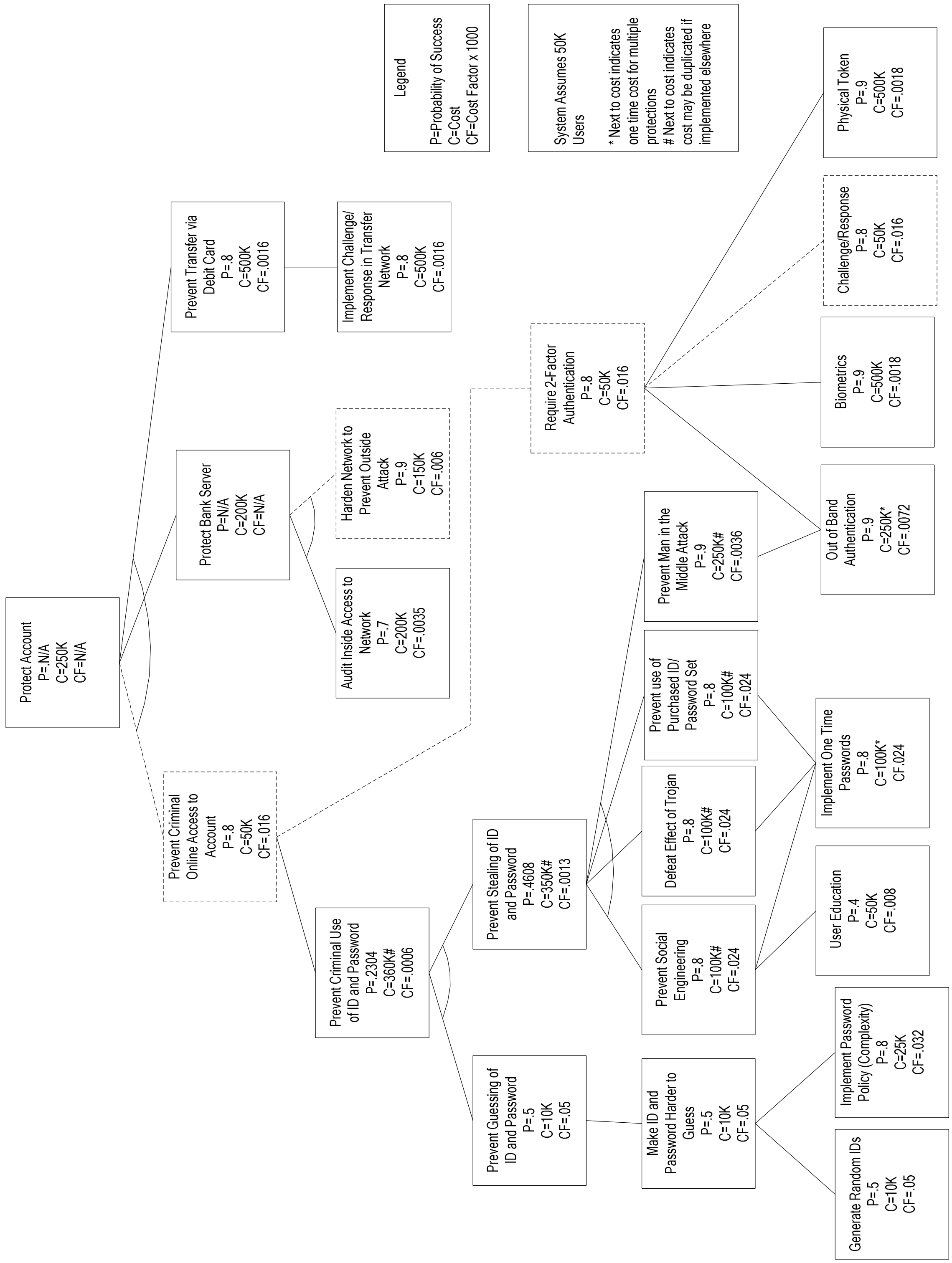

Figure 4.12: Online Banking Protection Tree Using $200K in Protection Resources.

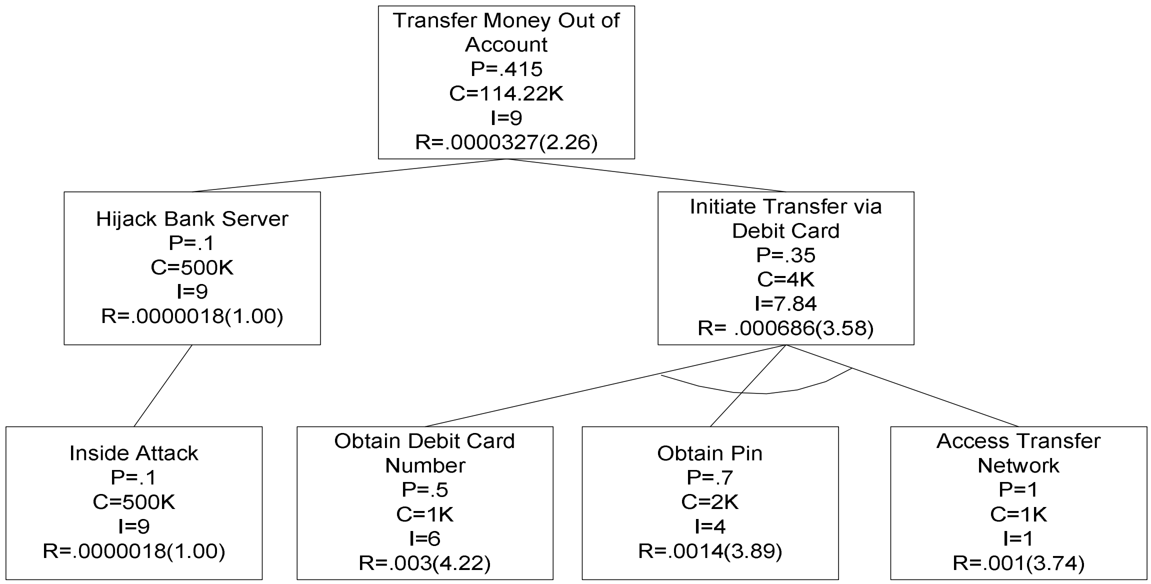

Figure 4.13: Attack Tree after Limited Protection.

protect customers' accounts [FFI06]. The risk analysis shows why multi-factor authentication is recommended. It is cost effective and provides reasonable security.

When limited protection resources are available, the protection tree method of risk analysis determines where future resources should be focused if and when they become available. This also helps determine which protections to implement first in a multi-stage implementation plan.

Attack and protection trees also identify which protections may not be necessary. By modeling an attacker and seeing how the protections relate to each other, it is possible to identify protections that do not add significantly to the protection scheme and yet add considerable cost. This is not always intuitive and the protection tree provides a methodical way to identify this situation.

4.3.1.5 Online Banking Application Summary. This section implements attack and protection trees in the security analysis of an online banking system. Metrics

specific to this example are also defined and equations to propagate the metrics up the trees are developed. The utility of the attack and protections trees is fully realized in an environment of limited protection resources and aids the decision makers of a security system in choosing the most cost effective protections.

4.4 Mobile Ad Hoc Networks

Mobile Ad Hoc Networks (MANETs) are becoming increasingly important for applications in hostile environments such as military, homeland security, and disaster response. MANETs are desirable mainly due to their node mobility and lack of fixed infrastructure. For these same characteristics, the security of these networks can be complex. The limited computing power, bandwidth, and energy resources of the individual nodes also complicate the security issues in MANETs.

Attack and protection trees are used to identify specific vulnerabilities as well as the protections needed to mitigate these vulnerabilities in a MANET. The use of protection trees determines which security measures should be implemented for a MANET when resources are limited by identifying which are the most efficient ones. Because efficiency can be measured in both cost as well as performance, a tradeoff analysis between these factors needs to be conducted before decisions about implementation are made. This is illustrated using a sensitivity analysis of the actual tradeoff factor. This analysis can be presented to a decision-maker who can ultimately decide the relative importance of these competing factors in a specific MANET. Once the weighting factor is determined, specific protection solutions are selected resulting in the most efficient protection scheme.

4.4.1 Motivation MANET Application. MANETs are being utilized in many applications such as military, homeland security, and disaster response primarily due to their flexibility, mobility, and lack of fixed infrastructure. For these same reasons, security in MANETs is a challenge and much different than security in wired networks. To determine vulnerabilities and countermeasures in MANET security, attack and protection trees are implemented. A precise methodology is developed that ensures the most cost effective protections are chosen for implementation.

Due to limited computing power of some nodes, as well as possible power consumption issues, the performance cost of selected protections can be significant. To balance the financial costs against the performance costs, a sensitivity analysis is conducted and presented to the decision-maker. With this information, the decision-maker can choose a tradeoff cost factor that can be used to transform the multi- objective performance versus cost problem into a single-objective combined cost problem.

As previously stated, there are competing cost factors in MANETs. There are three ways to deal with this issue. The first is to optimize security while limiting the financial cost, ignoring the performance cost. This often results in a system that may be very secure bt unusable due to the weak performance characteristics of the system. The second way to deal with the problem is to optimize security while minimizing the performance cost. Again, this results in a secure system but often at an unrealistically high financial cost. The best solution usually involves optimizing security while attempting to minimize both financial cost and performance cost. This creates a multi-objective problem and a set of possible solutions that the decision-maker can choose from

depending upon the weights of importance for cost versus performance. This set of solutions is called a Pareto Front (Figure 4.14) where each solution is equally good from a security standpoint. A specific solution is selected from the set only after a weighting factor between the competing objectives is determined. The weighting factor determines the amount of tradeoff between the financial cost and the performance cost of the system while maintaining the specified security level of the system. For example, a system may be able to maintain a protection level with a probability of success of 80% using many different types of protection schemes. One scheme may have a performance cost of 50% and a financial cost of $10,000 while another may have a lower performance cost of 20% and a higher financial cost of $40,000. If the weighting factor is determined to be $1,000 per percent of performance cost then these two solutions are determined to be equally good because they both provide an 80% probability of protection with an equal total cost of $60,000.

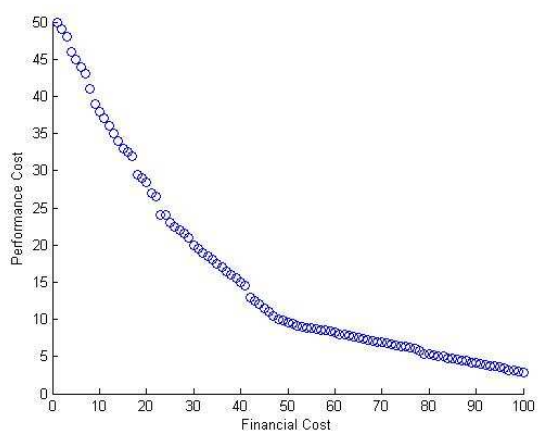

Figure 4.14: Pareto Front of Security Solutions for a System. This graph illustrates that a similar level of security may be possible while trading financial cost for performance cost.

Although the specific metrics for the MANET attack and protection trees discussed here are probability, cost, impact, and risk, other systems may have additional metrics. These may include the probability of detection, technical skill required, inconvenience to user, and damage cost to system. The idea is to capture the characteristics of the specific system being analyzed and consider how it is affected by the attacks and protections.

4.4.2 MANET Application. To illustrate how a security analysis is conducted on a MANET, a simple attack and protection tree is constructed for a hypothetical network. After establishing the attacker's root node as attacking the MANET, a decomposition can begin. It is determined that there are two fundamental ways to attack a MANET. Either an internal node can be compromised and used for malicious purposes or an external intruder can infiltrate the MANET using his own equipment. To simplify the analysis, only those attacks that originate from an external source are considered.

The three types of attacks on security can be decomposed in terms of confidentiality, integrity, and availability. To model these types of attacks, the attacker's ability to gain information on the network as a breach of confidentiality, deception as an attack on integrity, and denial of service as an attack on availability are used. These attacks are further decomposed and the complete attack tree is shown in Figure 4.15. The specific attacker actions are not specified to keep the example simple. If they were specified, an example attacker action for one of the denial of service attacks might be to abuse the protocols by sending many route request (RREQ) messages. The other general attack leaf nodes would also need to be specified in an actual attack tree.

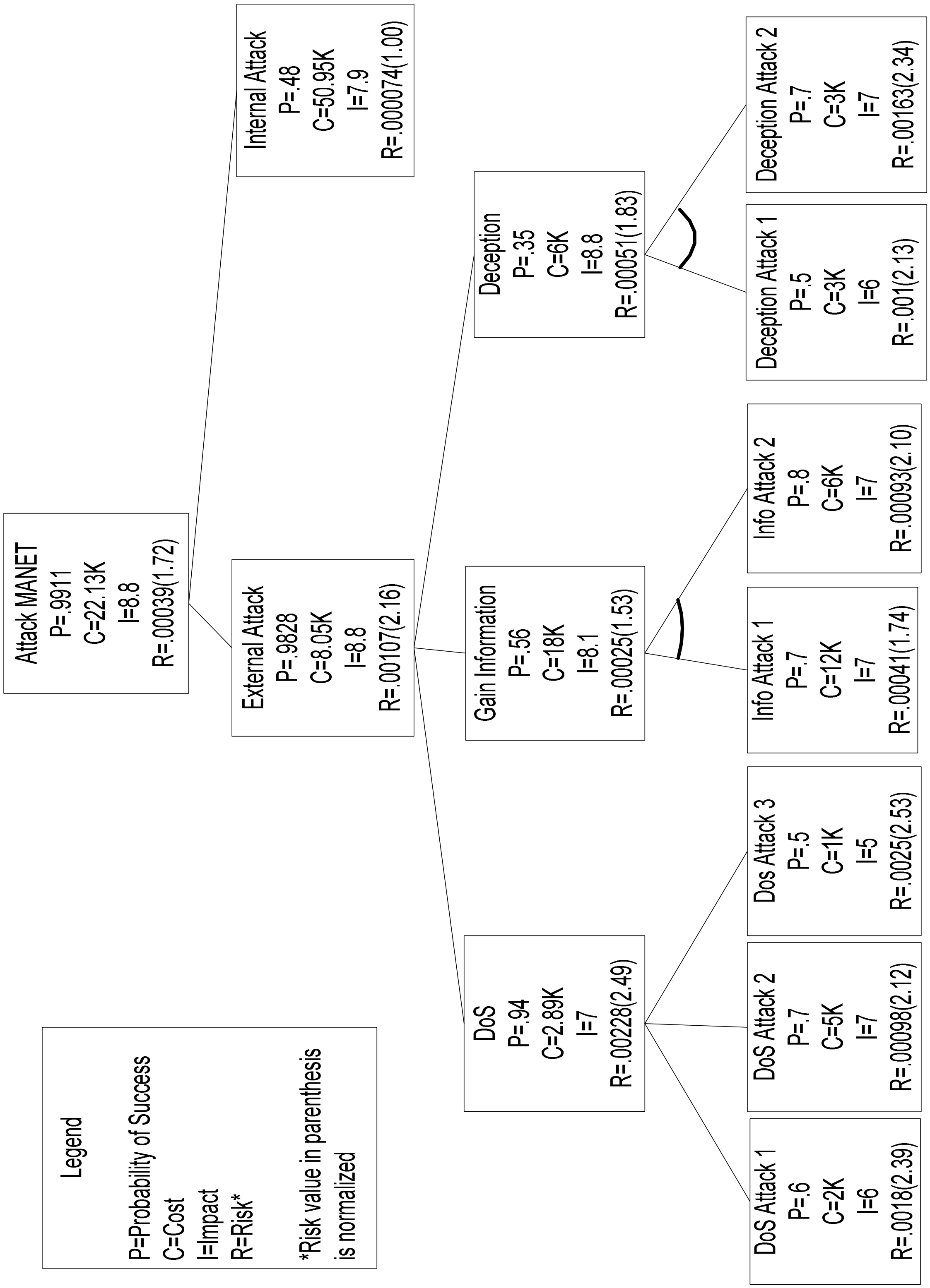

Figure 4.15: MANET Attack Tree.

Notional metrics are assigned to the leaf nodes and propagated up to the root node using the rules developed previously. Note that the numerical values in this attack tree are notional and are used simply to demonstrate the analysis methodology.

After the attack tree is constructed, the associated protection tree is developed. Like the example attack tree, the protection leaf nodes are generalized to simplify the explanation of the security analysis process. On the first iteration, the maximum protection level is determined. This protection level assumes that every possible protection is implemented regardless of cost or performance. This protection tree is shown in Figure 4.16. The second iteration of the protection tree determines the most efficient protections to implement when choosing between protections related by an OR node. This ensures every attack is protected to some level but is usually less than the maximum protection level determined by the first iteration. To determine which nodes to select, the tradeoff weighting factor between financial cost and performance must first be determined.

At this stage in the methodology, a sensitivity analysis is done on the weighting factor to present to the decision maker. To do this, the sets of nodes are determined that are related by an OR relationship and offer some level of protection against every attack. In Figure 4.16, nodes must be selected from the *protect against info attack* nodes and from the *protect against deception attack* nodes. There are four possible solutions that can propagate up and the total cost factor at the *prevent external attack* node can be measured while varying the weighting factor.

To generate the results shown in Figures 4.17 and 4.18, the *prevent external*

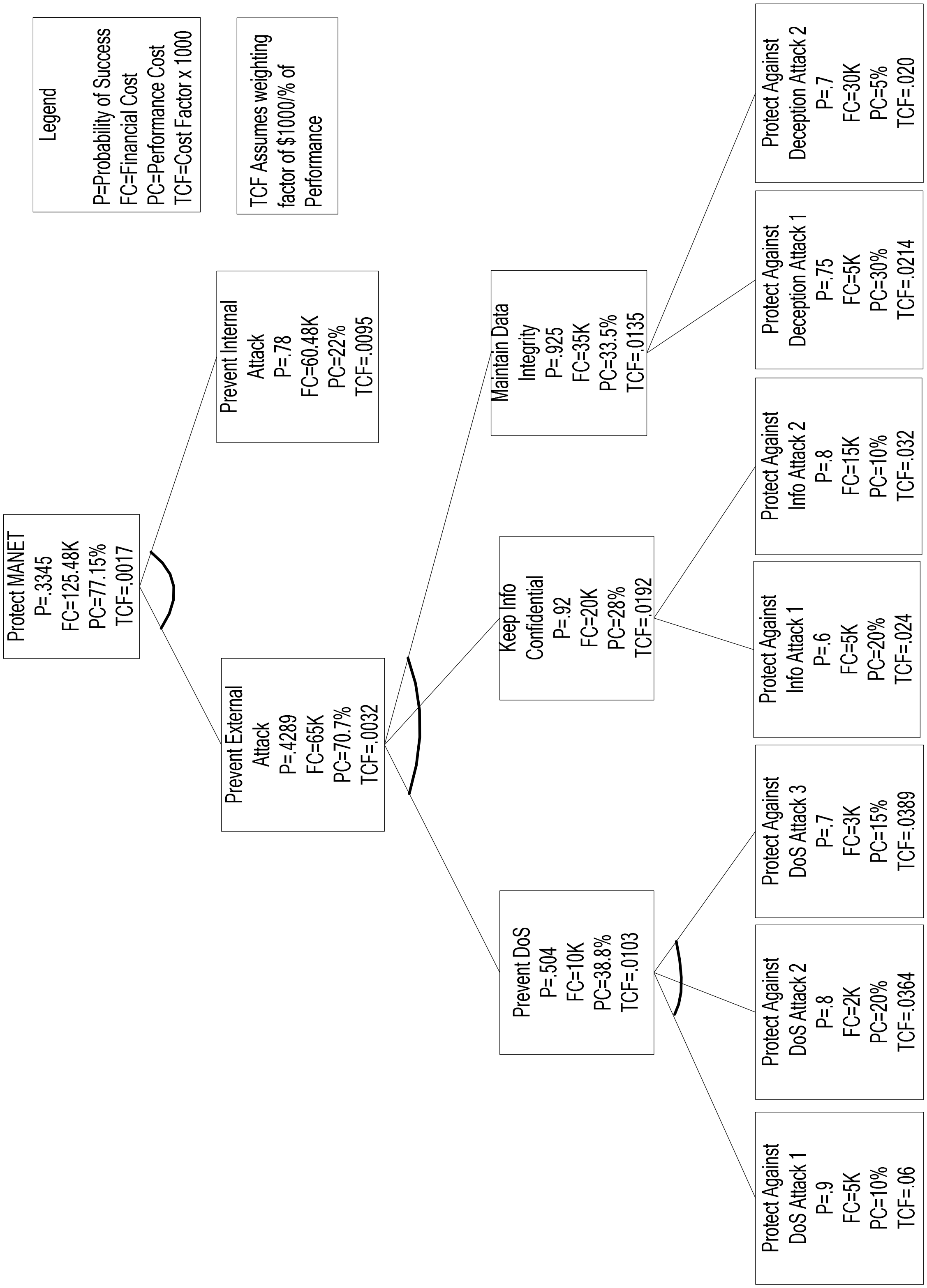
Legend
P=Probability of Success
FC=Financial Cost
PC=Performance Cost
TCF=Cost Factor x 1000
TCF Assumes weighting factor of $1000/% of Performance
Protect MANET P=.3345 FC=125.48K PC=77.15% TCF=.0017
Prevent Internal Attack P=.78 FC=60.48K PC=22% TCF=.0095
Prevent External Attack P=.4289 FC=65K PC=70.7% TCF=.0032
Prevent DoS P=.504 FC=10K PC=38.8% TCF=.0103
Keep Info Confidential P=.92 FC=20K PC=28% TCF=.0192
Maintain Data Integrity P=.925 FC=35K PC=33.5% TCF=.0135
Protect Against DoS Attack 1 P=.9 FC=5K PC=10% TCF=.06
Protect Against DoS Attack 2 P=.8 FC=2K PC=20% TCF=.0364
Protect Against DoS Attack 3 P=.7 FC=3K PC=15% TCF=.0389
Protect Against Info Attack 1 P=.6 FC=5K PC=20% TCF=.024
Protect Against Info Attack 2 P=.8 FC=15K PC=10% TCF=.032
Protect Against Deception Attack 1 P=.75 FC=5K PC=30% TCF=.0214
Protect Against Deception Attack 2 P=.7 FC=30K PC=5% TCF=.020

Figure 4.16: MANET Protection Tree (First iteration).

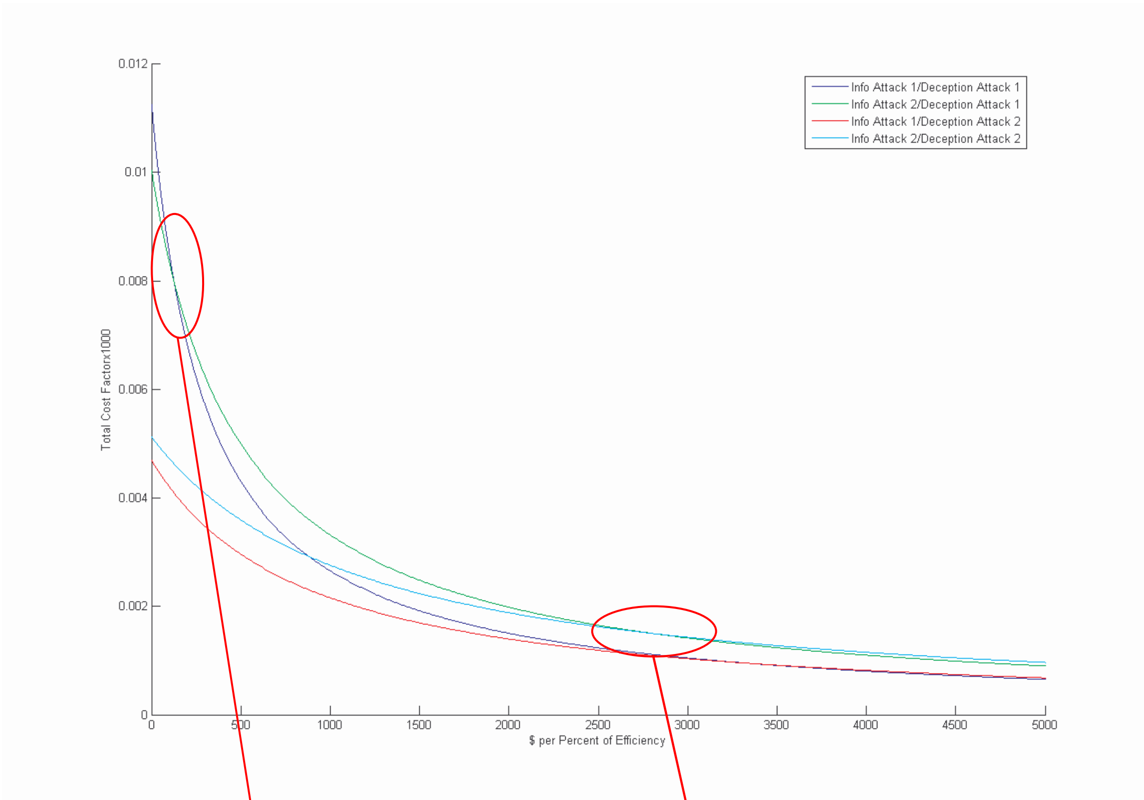

Figure 4.17: Sensitivity Analysis of Weighting Factor. Areas With Red Circles Are Expanded In Figure 4.18**.**

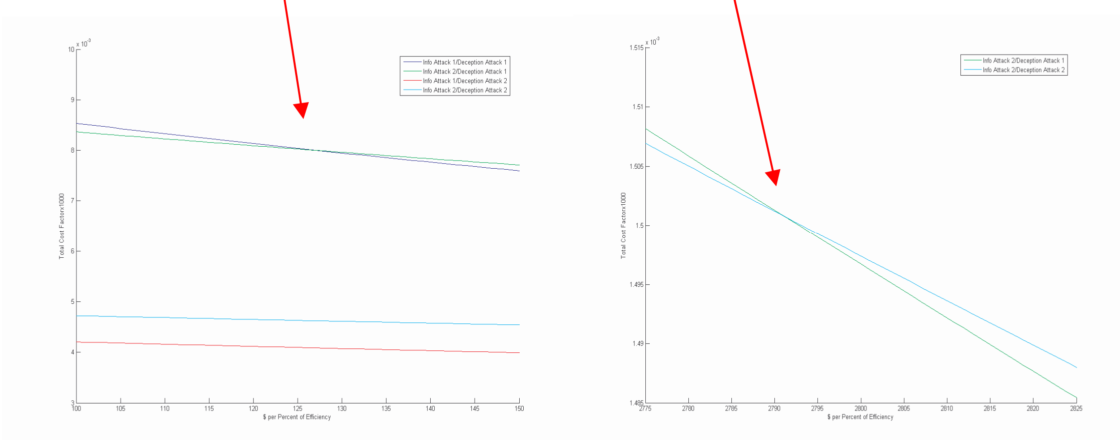

Figure 4.18: Expanded Views of Critical Points in Sensitivity Analysis.

attack node is calculated by varying the weighting factor. To plot the *info attack 2/deception attack 1* line, the associated nodes are propagated up and combined with the *prevent DoS* AND node. Using the prorogation rules from Table 3.3, *probability*=.4289, *financial cost*=65K, and *performance cost*=70.7%. To obtain the *total cost factor* plot, the *weighting factor* is varied between $0 and $5000 and applied to the *tcf* equation (3.3) which becomes $.4289/(65 + weight \times 70.7)$. The same process is used to obtain the remaining three plot lines to complete the sensitivity analysis.

After analyzing Figures 4.17 and 4.18, it is determined a weighting factor of less than $127 per percent of performance means the most efficient protection to use is *protect against info action 1* and *protect against deception action 1*, since the financial costs are low for these protections and performance costs are valued relatively low at this weighting factor. From $127 to about $2790 per percentage of performance, the most efficient set of protections is *protect against info action 2* and *protect against deception attack 1*. Above a weighting factor of $2790, *protect against info action 2* and *protect against deception action 2* becomes the most efficient set of protections. This is because these protections have relatively low performance costs and performance is more valued at these relatively high weighting factors.

Presenting the preceding sensitivity analysis to the decision maker, it is assumed he is keeps the initial weighting factor of $1000 per percentage of performance. After selecting the appropriate protections and recalculating the protection tree shown in Figure 4.19, the final metrics for the root protection node are; *probability*=.2359, *financial cost*=90.48K, *performance cost*=69.92%, and *total cost factor*=.0015.

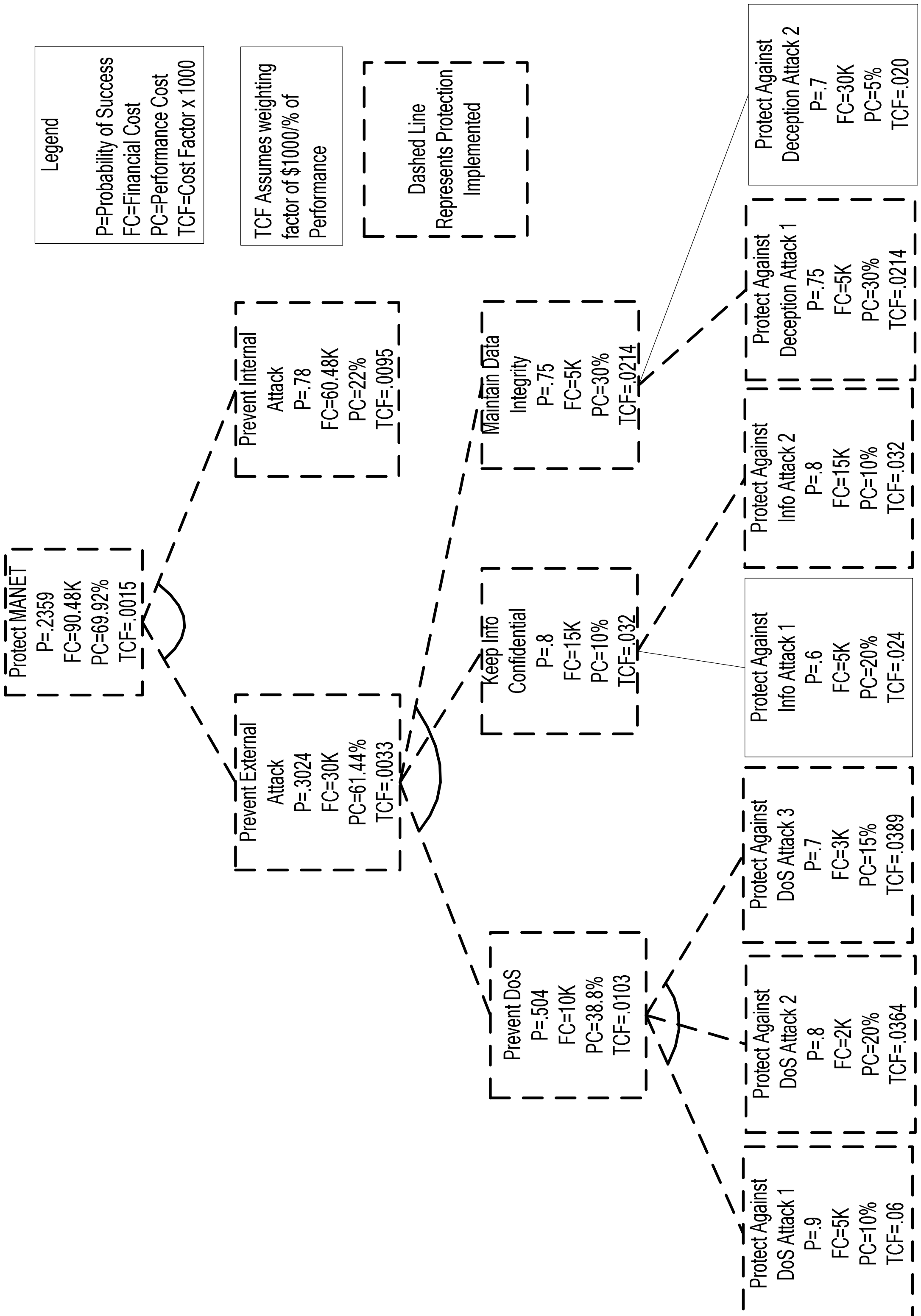

Figure 4.19: Second Iteration MANET Protection Tree.

4.4.3 Results and Analysis. After conducting the preceding security analysis on the MANET, the most efficient protection scheme is determined. If resources are further constrained either in financial costs or performance penalties, then protections are chosen, using the most efficient protections first until available resources are exhausted. This may result in a partial protection scheme that does not mitigate every attack. If this is the case, final metrics are not always possible to calculate for the protection tree. To measure the effectiveness of the protection scheme, the original attack tree can be pruned by removing the attacks that are mitigated. The attack tree is then recalculated to measure the effectiveness of the protections. The effectiveness is specifically measured in a reduction of the attacker's probability of success, an increase of the attacker's cost, and a reduction in the impact on the system.

4.4.4 MANET Application Summary. Attack and protection trees provide an analyst with the tools to properly conduct a security analysis on a system such as a MANET. The methodology presented allows an analyst to model the tradeoffs between system cost and performance while trying to optimize security. Using these models, the analyst can work with a decision-maker to establish the appropriate tradeoff between cost and performance to achieve the most efficient security scheme while staying within the limits of available protection resources.

4.5 Research on Specific Vulnerability Detectors and Protections

As part of this research effort, studies are conducted in areas that are employed in actual attack and protection trees. One area of research is placing virus protections in legitimate DoD applications to protect them from reverse engineering. After developing

general protection ideas from the viruses, a single protection mechanism, metamorphism, is developed further, implemented, and tested. The use of an artificial immune system based intrusion detection system for mobile ad hoc networks is also explored as an application. The final area of application develops a detector for metamorphic viruses using an algorithm based on the human immune system's use of reverse transcription. This algorithm is tested on a variety of test functions with promising results. The algorithm is also used by another researcher in a test intrusion detection system with excellent results.

4.5.1 Virus Protections as Legitimate Software Protections. This research investigates the protection techniques and methods used by malicious code writers. These protections are categorized and a brief overview for each of the protections is in Chapter II. This section describes ways to use these protections in legitimate software applications. These protections can be incorporated into a library of software protections.

The motivation behind this effort is to tap into the vast knowledge of computer architectures, operating systems, and software exploitation possessed by the virus authors. Many of these exploits and "tricks" are undocumented. It is possible that legitimate software applications can apply many of the same methods for protection.

Malware protective mechanisms can provide the Software Protection (SP) community with new ideas and methods for protecting sensitive software applications. The primary motivation of the malware author and the SP professional is ironically similar: both want to extend the time and resources required to reverse engineer their respective pieces of software. In the case of the malware author, the intent is to

maximize the available propagation time for pride and ego—not to protect legitimate software.

4.5.1.1 Software Protection Applications for Tunneling. Consider a scenario, where a process won't allow an application to run unless it noticed a specific indicator in the master boot record (MBR). This indicator could be a checksum of a process loaded early in the boot sequence and translated into a key that decrypts the application. This process could "hook" certain system calls that make the application undetectable so an adversary cannot view it. If the application runs on a trusted computer, the process loads and the application decrypts and runs. If the software runs on an untrusted platform, the process is not present and therefore the key for the decryption is incorrect and the application does not run as intended. It might be desirable to let the application execute but run incorrectly to fool an adversary.

4.5.1.2 Software Protection Applications for Encryption. The SP community already makes wide use of encryption as one of the primary means of protecting an application from unauthorized execution or reverse-engineering. By studying viruses that use encryption, there may be novel techniques that can be used depending on the application and the operating environment. One note of caution when using encryption is to avoid a false sense of security. If a key can be obtained using social engineering or by foiling poor storage policies, no matter how strong the encryption is, it becomes worthless.

4.5.1.3 Software Protection Applications for Stealth. Stealth techniques can be applied when creating watchdog applications that guard a critical application. The

presence of these guards remains hidden so an attacker cannot bypass them. Stealth techniques are also employed to hide other protections that are used in an application. For example, a single application may have multiple versions with a different set of protections in each version. To keep an attacker from knowing which version is running, stealth techniques display the size of the program as the same between different versions

4.5.1.4 Software Protection Applications for Anti-debugging. From a SP viewpoint, anti-debugging hinders an adversary from fully understanding the protected software by limiting the types of software tools normally used in reverse engineering such as OllyDbg and Soft Ice.

4.5.1.5 Software Protection Applications for Anti-heuristics. Anti-heuristic techniques can fool scanners that look for the existence of certain protections. Not allowing an adversary to easily find out which protections the application is employing, increases the time it takes to reverse engineer the application.

4.5.1.6 Software Protection Applications for Anti-goat Armoring. Anti-goat technologies apply to the SP community as well. The use of these tactics prevents the protected application from "leaking" too much information. For instance, if the protected application's function is to encrypt a particular set of data (such as database records), an attacker can employ a "known plaintext" attack against the application. If the program does not perform any kind of anti-goat checks, then the attacker can use the program to determine the algorithm used as well as the encryption key.

4.5.1.7 Software Protection Applications for Anti-emulation Armoring. In the SP realm, such anti-emulation techniques are useful for general purpose obfuscation. In

fact, many standard obfuscation tactics would likely perform similar functions. Limiting the attacker's use of a particular set of tools can extend the timeline required to successfully reverse engineer the protected software.

4.5.1.8 Software Protection Applications for Retrovirus Tactics. Retrovirus techniques can ensure that critical applications are only used on systems that do not utilize tools used in detection and reverse engineering. If these are detected by the application, the application can attack the tools used in reverse engineering or damage the system that it is running on. In a less harsh response, the application can disable scanners or some other type of detection system. The extent of the response relies on the sensitivity of the application and liability issues.

4.5.1.9 Software Protection Applications for Morphing Viruses. Morphing has perhaps the most significant applications to the SP community—especially when considered in conjunction with other protection mechanisms. Metamorphism can provide additional obfuscation *potency*, or power to confuse, versus humans and *resiliency*, or ability to endure, against automated deobfuscation tools.

Collberg et al. classify obfuscation tactics with respect to their potency against humans, resilience against automated deobfuscators, stealth of obfuscations versus original code, and performance cost. Obfuscators incorporate these protective measures into source code and binary executables via layout, data, control, and preventive transformations [CoT97].

Combined with obfuscation tactics, metamorphic techniques are a "force multiplier." First, automated deobfuscators have a similar level of difficulty as anti-virus

companies have with morphing viruses (resilience). Deobfuscators have to successfully leap a significant technology barrier—the development of a successful heuristic scanner. The SP community should consider the short-term costs associated with the development of a good heuristic scanner for deobfuscating. Such a scanner is likely be very slow, which adds more long-term costs (specifically time) to the adversary's future reverse engineering efforts.

Second, the obfuscation techniques will likely become even more effective against humans (potency). If the code changes during each execution, then the difficulty of the reverse-engineering effort can quickly overwhelm human adversaries. Manually following metamorphism appears problematic to say the least. Perhaps the best methodology against metamorphism is to understand the basic functionality of a particular section of code and not reexamine the section, because reexamining the code would add "needless" confusion. However, during the next execution, if the metamorphic engine swaps the order of subroutines such that it does not overly resemble their previous forms, it might prove difficult to determine where the reverse engineering effort left off.

Additionally, a symbiotic relationship can exist between metamorphism and anti-debugging. One of the first tasks for the reverse engineer is likely to be disabling of the anti-debugging protections in the code. However, if the anti-debugging protections keep changing due to metamorphosis, then the overall protection will increase potency and resilience.

4.5.1.10 Taxonomy of Virus Protections Summary. Malicious software

developers continue to challenge the protection community with advanced techniques to avoid detection and protect of their code. This section discusses many of these techniques as well as possible uses for them in the non-malicious software arena. When extracted from the viruses and understood, these techniques can be used for legitimate purposes in critical software application protection mechanisms.

4.5.2 Metamorphism as a Software Protection This research further develops one of the virus protections, metamorphism, to find out how well it can work as a protection in a legitimate software application. Although the main impetus for this protection in malware is to avoid detection from anti-virus signature scanners by changing the program's form, certain metamorphism techniques also serve as anti-disassembler and anti-debugger protections. For example, opcode shifting is a metamorphic technique used to confuse program disassembly. This research assesses the performance overhead of a simple opcode-shifting metamorphic engine and evaluates the instruction reach of this particular metamorphic transform.

4.5.2.1 Motivation for Metamorphism as a Software Protection. Protection of sensitive government software or a software vendor's intellectual property is a concern for information assurance professionals. Whenever the software security community introduces a new protection, a game of cat and mouse often ensues where code crackers immediately set out to defeat it. In this section, the performance overhead and potential effectiveness of existing protection from malware in non-malicious applications is examined.

4.5.2.1.1 Research Goal. This research examines the performance overhead

of one type of metamorphic transform, namely instruction opcode shifting. In addition, the instruction “reach” of such a transform is analyzed to provide the software protection community with an estimate of the number of shifts required to protect a section of code.

4.5.2.2. Additional Background on Software Metamorphism. Metamorphosis is “a marked change in appearance, character, condition, or function”. Literally translated from its Greek roots meta- and –morph, it means “many forms or shapes” [Ame00]. In software, one could easily argue that metamorphism refers to any self-modifying code from a purist perspective. As a software protection, metamorphism refers to any alteration of the program structure, execution flow, function, or data that serves as a protective measure.

Although virus writers draw much of the attention today, other applications of self-modifying code exist as well. Giffin, et al. proposes using self-modifying code as a means of strengthening software self-checksumming [GiC05]. Yip and Zhao advocate using metamorphism for enhancing the "registration key" protection for software [YiZ04]. Historically, practitioners use self-modifying code to save memory space and to supplement limited instruction sets. Real-time graphics developers use self-modifying code to generate code at runtime [Wik05].

4.5.2.2.1 Metamorphism as a Software Protection Mechanism. In general, software security practitioners do not consider metamorphism as a legitimate protective measure, perhaps based on its association with malicious software or ideological arguments against self-modifying code. However, metamorphic techniques offer software more protection against reverse engineering by providing additional anti-

disassembly and anti-debugging protections. The potential level of protection obtainable by metamorphism justifies determining both its performance overhead and effectiveness as a legitimate protective measure.

4.5.2.2.2 Oligomorphism and Polymorphism in Viruses. Metamorphism in malware is the result of an ongoing progression of ideas to avoid signature detection and ultimately improving malware resilience against automated detection tools. The increased resilience allows malware programs to propagate freely for longer periods of time since anti-virus researchers have to develop signatures or other detection methods to effectively detect all forms of the malware program. Metamorphism has potential applications other than merely avoiding detection, its chief advantage for malware. Certain metamorphic transforms can serve as an anti-disassembly and anti-debugging protection as well.

Viruses primarily use morphing techniques to avoid signature detection by anti-virus products. Two common morphing techniques are oligomorphism and polymorphism. The distinguishing characteristic between the two is one of scale—namely, the scale of how many different variants they can produce. Oligomorphic viruses generate a few variants while polymorphic viruses generate many variants during propagation.

Figure 4.20 shows a general form used by some encrypted viruses. Each time these viruses replicate (or propagate), they encrypt their main bodies with a different encryption key storing the new key in the decryptor. This practice makes generating a virus signature, based on any component of the main body, a fruitless effort. Decrypting

the main body immediately prior to execution reduces the virus' exposure to memory signature scans based on the decrypted main body. Faced with this anatomy, anti-virus researchers develop signatures based on the relatively small "decryptor" portions of the virus code [Szo05].

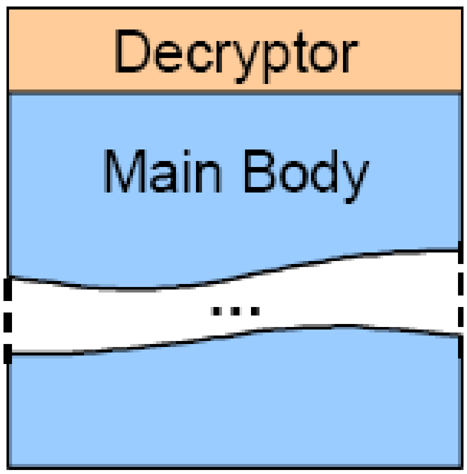

Figure 4.20: Anatomy of a Simple Encrypted Virus Program.

Virus writers make this task more difficult by using oligomorphic and polymorphic techniques, which generate multiple variants of the already small virus decryptor. During virus analysis, researchers must be certain to generate signatures that capture all possible variants. Generating such signatures is not an easy task considering that a polymorphic virus can generate millions of different decryptor forms. Some simple virus variants are mutations that incorporate junk instructions and change the order of independent instructions.

4.5.2.2.3 Metamorphism in Viruses. Following the simple virus variants, the next generations of viruses introduce metamorphic viruses. These malicious programs do not bother with encryption (and therefore do not need a decryptor) unlike their close

relatives, oligomorphic and polymorphic viruses. Rather than encrypting their main bodies, metamorphic viruses change their shape (or appearance) so dramatically that they are simply unrecognizable.

Methods to change form include insertion of junk instructions, instruction opcode shifting, equivalent instruction substitution, independent instruction reordering, subroutine reordering, and register substitutions. These types of viruses can produce an extremely large number of mutations. Furthermore, the parent normally avoids producing an offspring that bears too much of a "family" resemblance [Szo05]. Other metamorphic transformations exist as well, such as reversing conditions in a conditional branch [Eil05].

4.5.2.3. Experimental Design. Two independent experiments test the performance and instruction reach of the specified metamorphic transform. The test application for these experiments is a benchmark program included in the NIST project called SciMark 2.0, which program measures performance in scientific and engineering applications [PoM04].

Each experiment provides the appropriate data to simplify analysis. For instance, examining the instruction reach, defined as the number of instructions the debugger cannot resolve, of the transform does not require measuring any type of performance metric. Furthermore, randomly determining if a metamorphic transform should occur is also unnecessary for the instruction reach test. On the contrary, the performance experiment requires consideration of a random number generator to determine if a metamorphosis should occur.

The metamorphic transform of instruction opcode shifting inserts code into an application that performs a jump over carefully chosen data bytes as shown in Figure 4.21. When a disassembling application finds this sequence of instructions and data bytes, it must decide how to disassemble them correctly.

```
00402B7C  EB 02                     JMP SHORT a.00402B80
00402B7E  81                        DB 81
00402B7F  80                        DB 80
00402B80  C740 04 00000000          MOV DWORD PTR DS:[EAX+4],0
00402B87  833D 38604000 00          CMP DWORD PTR DS:[406038],0
00402B8E  75 16                     JNZ SHORT a.00402BA6
00402B90  A1 20604000               MOV EAX,DWORD PTR DS:[406020]
```

Figure 4.21: Two Bytes of Data (0x81 and 0x80) set up an 8-byte Opcode Shift.

The basic problem is that a carefully selected opcode shift presents the disassembler with two viable options. The disassembler could consider the two bytes as data bytes or as an instruction opcode with a possible prefix. Figure 4.22 shows an alternative disassembly with subsequent garbled instructions until the instruction at address 0x00402b90.

```
00402B7C  EB 02                     JMP SHORT a.00402B80
00402B7E  8180 C7400400 00000083    ADD DWORD PTR DS:[EAX+440C7],83000000
00402B88  3D 38604000               CMP EAX,a.00406038
00402B8D  0075 16                   ADD BYTE PTR SS:[EBP+16],DH
00402B90  A1 20604000               MOV EAX,DWORD PTR DS:[406020]
```

Figure 4.22: A Successful 8-Byte Instruction Opcode Shift.

Finding a method of inserting the required bytes into the available space for these opcode shifts is not a trivial problem. Shifting all subsequent instructions is problematic, because the program must resolve any absolute and relative addressing issues (for subsequent and preceding instructions) and may have to modify its own segment table to

accommodate the increased code size. For simplicity, "morph point" macros are inserted into the target source code that performs a simple conditional test that short-circuits. The disadvantage to such an approach is that the program tests the bogus condition every single time it executes. It is assumed that the cost of performing an actual shift of the instructions and addressing all the resultant issues for each insertion is significantly more costly than the time spent checking the bogus condition for each morph point.

For these experiments, the morph points are inserted into the test application before different types of instructions, such as simple variable assignments, array element assignments, and conditional checks. At least 10 assembly instructions are between each morph point, which allows the transform to morph at least 10 instructions. Preliminary tests show that morph point effects did not reach past this number of instructions.

4.5.2.3.1 Performance Experiment. The performance experiment assesses the performance of both the metamorphic "engine" and the modified code. The test application first uses a high-performance timer to determine the time elapsed between events. The metamorphic engine then examines a list of pre-initialized morph points, randomly determines if it should modify the morph point, and modifies the morph point accordingly. The metamorphic engine can modify a particular morph point multiple times with a random opcode shift. Of particular note, the decisions to morph are independent—even for subsequent iterations. This experiment does not involve any debuggers, because it is performance-based and measuring its execution performance inside the context of a debugger is meaningless.

This experiment considers two factors, the compiler and the host program. There

are two levels chosen for the compiler factor: the Cygwin GCC C compiler and the Microsoft Visual Studio .NET 2003 C compiler. The five applications comprising the NIST SciMark 2.0 benchmark suite serve as the levels for the host program factor. The factors are chosen to observe the performance impacts due to how the compiler generates the executable as well as the instruction mix of the application.

4.5.2.3.2 Instruction Reach Experiment. The instruction reach experiment assesses how many instructions the opcode shift can manipulate with popular debuggers. This experiment executes code in a debugger and captures the resulting disassembly to an output file. After generating the disassembly, the output is compared with the baseline assembly to determine how many subsequent instructions the opcode shift mangles.

Numerous methods for shifting opcodes exist. The simple approach of using various opcodes (and prefixes) for the add instruction are used. By manipulating the opcode to adjust the displacement and immediate fields, the metamorphic engine is capable of generating shifts of 1, 2, 3, 4, 5, 6, and 8 bytes. Figure 4.23 shows a simplified version of the Intel Architecture 32-bit instruction format [Int05] which shows why the shifts do not include 7 bytes.

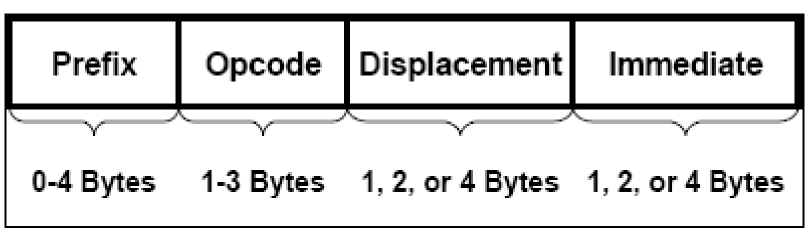

Figure 4.23: Simplified IA-32 Instruction Format.

This experiment considers four factors, the debugger, the opcode shift amount, the compiler, and the host program. The debugger factor has two levels, IDA Pro [Dat05]

and OllyDbg v1.10 [Yus05]. The levels for the opcode shift amount range from 1 to 8 bytes (excluding 7 bytes as described previously). The compiler levels are the Cygwin GCC C compiler and the Microsoft Visual Studio .NET 2003 C compiler. The levels for the host program are the five applications comprising the NIST SciMark 2.0 benchmark suite.

4.5.2.4. Results

4.5.2.4.1 Performance Experiment. The performance tests on all five benchmark applications with the GCC C compiler are conducted. This section describes the results from the completed GCC C compiler performance tests on the benchmark applications.

The performance tests measures the execution time of the baseline application and the morphed application. In addition, the experiment measures the performance of the initialized metamorphic engine for each metamorphosis. During test iterations, the metamorphic engine randomly considers each morph point in the test program and modifies the appropriate morph points. In this scheme, the program can modify itself in part or in its entirety during a single iteration.

The preliminary results for the performance tests show an overall 4.5% increase in execution time for the morphed programs when compared to their original forms. Several morph points are placed into each application's code for the metamorphic variants. The baseline contains no morph points. Table 4.1 shows the difference in performance between the morphed and original code. An interesting observation from the preliminary results is the fact that the metamorphic code with the fewest morph points, Application 3, actually executes faster in the test case than the baseline. The fact

that the application in question has such a small number of morph points also implies that its performance should be quite similar to the original program.

Table 4.1: Table of Execution Time Ratio between Morphed and Original Code.

	Execution Time Ratio	Number of Morph Points
App1	1.091	30
App2	1.004	8
App3	0.991	6
App4	1.086	16
App5	1.054	23
Average	**1.045**	**16.6**

Several other factors that contribute to the above performance results are considered. For instance, the number of times the morph point code executes should have high correlation with the decreased performance. Another factor is compiler optimization, which might contribute to the logical discrepancy for the third application in the above results.

The simple metamorphic engine is quite efficient in the experiments. Table 4.2 shows the summary of performance data for the metamorphic engine using 25 replications. The average execution time for considering each morph point is approximately 5 μs (microseconds). A 95% confidence interval for engine execution time for each application indicates an average execution time between 4 and 5 μs for each morph point.

4.5.2.4.2 Instruction Reach Experiment. During analysis, it is noticed that previous morph point spacing assumptions turn out to be false. During initial tests, a single morph point causes the debugger to not be able to resolve at most four instructions. From these initial results, the metamorphic experiments are set up with morph points at

least 10 instructions apart. However, during analysis, a single morph point manipulating up to 18 instructions is observed.

Table 4.2: Table of Metamorphic Engine Performance Results.

	Metamorphic Engine Execution (µs)	95% Confidence Interval	Morph Points	Average Time per Morph (µs)
App 1	136	(120,152)	30	5
App 2	33	(31,35)	8	5
App 3	26	(24,27)	6	5
App 4	67	(59,75)	16	5
App 5	108	(95,121)	23	5

Figure 4.24 shows the test results for the instruction reach experiment for the GCC C compiler used with OllyDbg. On average, the effective reach is approximately three or more instructions for most opcode shifts. However, in the tests involving 3-byte opcode shifts, the average reach is only slightly higher than a single instruction.

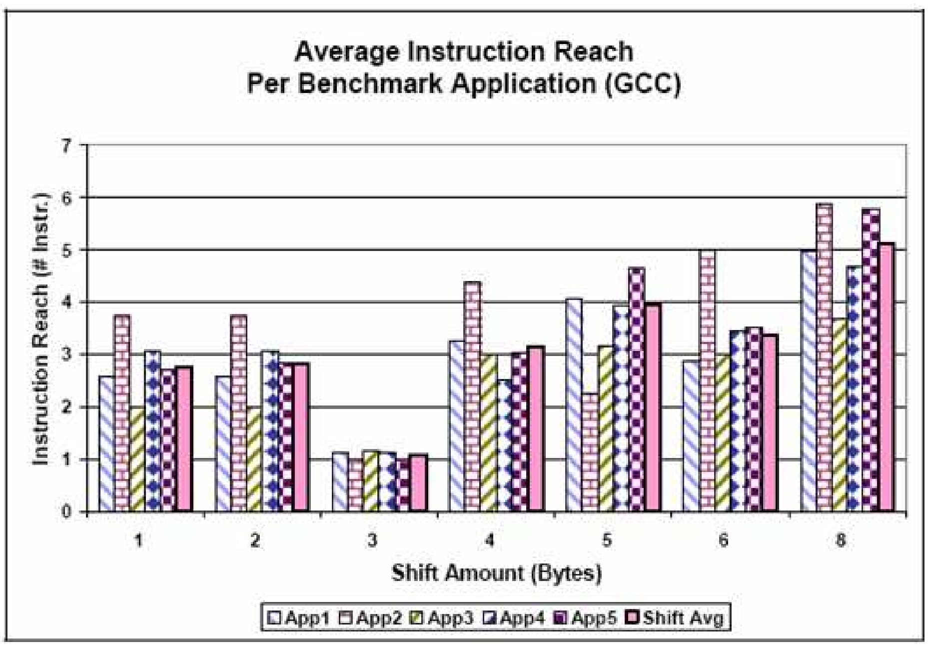

Figure 4.24: Average Instruction Reach Results.

This observation is interesting because the instruction reach for three-byte shifts is less than half the other test points. An analysis of the original codes' instruction sizes appears to explain this phenomenon. The majority of the instructions for all five of the benchmark programs are three bytes long as shown in Figure 4.25.

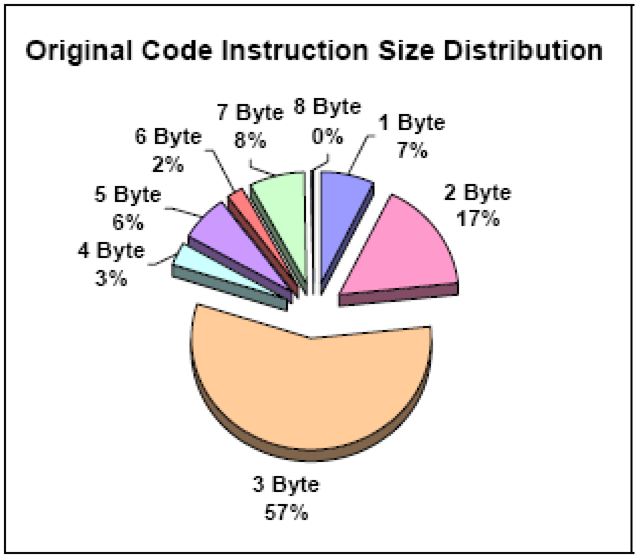

Figure 4.25: Overall Distribution of Instruction Sizes In the Original Five Benchmark Programs.

Taking this into account, the reduced reach of a three-byte shift seems intuitive. If the shift consumes the next three instruction bytes (and the next instruction is exactly three bytes), then the shift has an effective reach of one (only the next instruction). The fact that most instructions are three bytes long also explains the effectiveness of the one- and two-byte shifts.

Another interesting observation from Figures 4.24 and 4.25 is the effectiveness of the larger shifts. In theory, a four-byte shift (or larger) should completely consume the next instruction and mangle (if not absorb) the following instruction approximately 81%

of the time. This is an estimate of the probability that the next instruction is three bytes long or smaller. An analysis of the individual test data points supports this theory as well. Table 4.3 shows the observed percentage of times that the resulting reach exceeds a single instruction for 4-byte, 5-byte, 6-byte, and 8-byte shifts.

Table 4.3: Table of Observed Occurrences Where Large Shifts Absorb More Than One Instruction.

	4-Byte Shift	**5-Byte Shift**	**6-Byte Shift**	**8-Byte Shift**
Reach > 1	89	74	89	89
Samples	91	91	91	91
Percentage	**97.8%**	**81.3%**	**97.8%**	**97.8%**

4.5.2.4.3 OllyDbg Observations. Several other observations are made during development and experimentation. The observations lead to the conclusion that most debuggers are less effective when used to attack metamorphic code compared to static code analysis.

Many debuggers have elaborate user interfaces that simplify their use. OllyDbg has such an interface, but also has several limitations when used to disassemble or debug metamorphic code. After modifying its form, the morphed program hampers OllyDbg's ability to select a line of assembly instructions, which is a required feature for setting breakpoints correctly. However, OllyDbg does a good job of identifying the changed instructions via question marks in its symbology as shown in Figure 4.26. The position of the arrow shows where the user attempts to select an instruction. The highlight area shows the resulting selection, which clearly is not the user's intent. The question marks immediately to the left of the assembly bytecode indicate the changed instructions in the

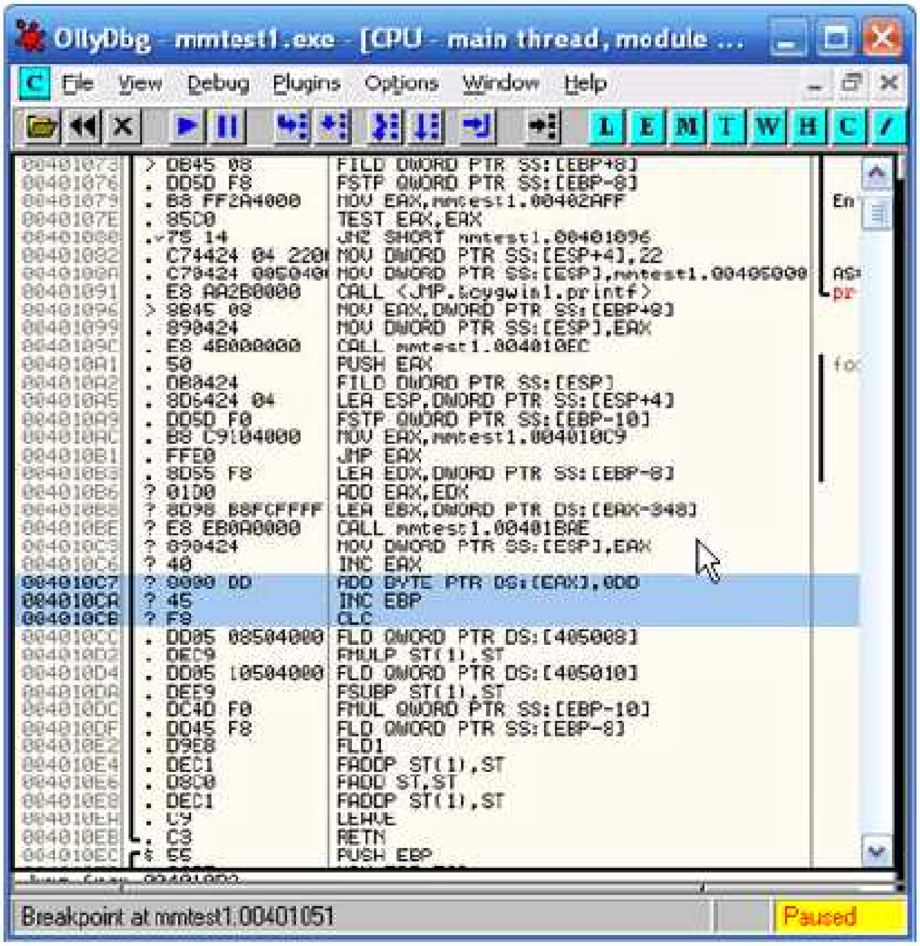

Figure 4.26: Snapshot of OllyDbg Showing The Morphed Instructions.

In addition, the instruction opcode shifting metamorphosis has another interesting anti-debugging effect. If a user places a software breakpoint in a morph point, the metamorphic engine randomly overwrites it. This is dependent on the metamorphic engine implementation, but it nonetheless reinforces this protection's effectiveness as an anti-debugging protection. If the user attempts to remove the breakpoint, OllyDbg identifies and describes the corruption of the breakpoint. However, OllyDbg gives no indication of the corrupt breakpoint until this user interaction.

OllyDbg behaves strangely when the user steps to the first unresolved instruction.

In the simpler opcode shifting transforms, OllyDbg correctly decodes the current hidden instruction, but as soon as the user steps to the next instruction or manipulates the window (i.e., scrolls, resizes, etc.); OllyDbg immediately hides the real instruction. With later advances of the metamorphic engine, OllyDbg never shows the correct decoding of the current instruction. This causes a mismatch between the instruction pointer and the address of the current highlighted instruction (assumed to be the current instruction).

These last two observations are interesting, because two basic functions a debugger provides is the ability to set breakpoints and to step through the code one instruction at a time. Effective instruction opcode shifting can cause debuggers to fail at both, which supports the claim that metamorphism can serve the software community as an anti-disassembly and anti-debugging protection.

4.5.2.5. Future Research and Recommendations. Metamorphism potentially offers unique advantages over traditional encryption because it is dynamic. On the other hand, metamorphism can increase the effectiveness of encryption by providing a random key and/or random decryption area to avoid memory dump attacks.

Other metamorphic transforms can augment instruction opcode shifting or other protection. For instance, some debuggers, such as OllyDbg, indicate the morphed instructions by some cue. Implementing subroutine reordering makes that cue less meaningful, because the vast majority of the instructions have the same cue (i.e., changes on top of changes). The indicator implies at least one change, but does not provide more information. Subroutine reordering might also foil hardware breakpoints, because the instruction where a user wants to pause might not exist at the same instruction address.

Furthermore, the desired breakpoint address might not coincide with the beginning of an instruction at all after a subroutine reordering, which can cause the debugged program to fail. Metamorphic variable redefinition is another transform that complicates the reversing process.

Testing the effectiveness of such measures against human reverse engineers would provide the software protection community with much needed data to quantify metamorphism's actual effectiveness. Such tests should use controlled experiments with an advanced metamorphic engine capable of morphing multiple instructions. The test programs could continuously morph themselves making them "moving targets" for reverse engineers with random groups of new opcode shifts to sort through each metamorphosis.

4.5.2.6. Summary of Metamorphism as a Software Protection Research. These preliminary experimental findings indicate the potential benefits of applying metamorphic opcode shifting to manipulate a sensitive program during execution and quantify representative performance overhead as well as the potential effectiveness of this technique in concrete terms.

Eilam alludes to opcode shifting as a popular anti-disassembly protection but also notes it causes little more than an annoyance to a reverser [Eil05]. An experienced attacker can easily write a script that removes these shifts. However, his comments likely reflect his opinion of static opcode shifting as opposed to dynamic shifts and raising the annoyance factor for the attacker is significant anyway. This type of self-modifying protection does not necessarily have to target a repeatable morph point either. Using the

same techniques, the metamorphic engine can reverse branches or even completely modify a sequence of instructions much like decryption.

Opcode shifting is a low cost protection in terms of difficulty to integrate and execution time. As previously stated, this type of self-modification does not need to target an opcode shift but any other opportune target.

Aside from the experimental findings, metamorphic transforms have utility as anti-debugging protections. In these experiments, metamorphism exposes a variety of debugger faults. These faults range from simple user interface bugs to debugger crashing.

4.5.3 Artificial Immune System (AIS) IDS for MANETS This section explores the use of anomaly detection as an IDS in mobile ad hoc networks. MANETs have inherent vulnerabilities based on their very nature. Use of an AIS has been proposed to provide robust and reliable misbehavior detection [SaB05]. This research builds upon that concept by framing the problem as a multi-objective problem attempting to balance the efficiency and effectiveness of the detection. After mapping the algorithm to a symbolic representation and discussing the design of the multi-objective AIS, testing results are discussed and a Pareto front of the results is presented. The results of this research can be incorporated into a vulnerability analysis of a MANET.

4.5.3.1 MANET Research Motivation. By their very nature, MANETs are especially vulnerable to malicious attacks. In a wired network, an attacker must either gain physical access to the network or pass through a predefined set of nodes that act as firewalls and/or gateways. In a wireless network, an attack can target any node and come

from anywhere. Because of this architectural difference, a wireless network must ensure every node is prepared for an attack and protect them accordingly [DaM05]. As wireless networks are rapidly developed, security is one of the greatest challenges for their implementation [KaG03].

An intrusion detection system for a mobile network can be designed with features similar to the human immune system (HIS). The HIS is modeled as an AIS with two objectives; to find intruders and act quickly. The two objectives can be restated as efficiency versus effectiveness. With multiple objectives the problem becomes one of a multi-objective artificial immune system (MOAIS).

4.5.3.2 Mapping to a Symbolic Representation. To map the problem of mobile network anomaly intrusions to an AIS domain, how each element of the problem domain maps to the AIS domain must be defined. The following mapping follows the convention described in [SaB03]:

Antigens represent the observed protocol events

Antibodies are created randomly and trained but the format matches that of antigens.

Self cells represent non malicious nodes

Nonself cells represent malicious nodes

Bone marrow is represented as a protected environment for learning with certified well behaving nodes

Clonal Selection represents the process of creating new antibodies from ones that are performing well. Poorly performing antibodies are replaced with mutated versions of

high affinity antibodies.

Protocol events are mapped to a finite set of primitives to form an alphabet (Table 4.4). The protocol events are recorded for a defined time and constrained to a maximum number of events [SaB03]. If the protocol streams are not confined to just a sampling, the sequences would quickly become too large to handle computationally. It is important to remember that the events that are recorded are only a representative sample. This means it could be possible to orchestrate a well crafted attack that can slip undetected between recorded events. To combat this, it is important to randomize the time intervals between events that are recorded. For the sake of simplicity, it is assumed that this is correctly done and it is infeasible to mask an attack by inserting it between recorded events.

Table 4.4: Alphabet of Primitives [SaB03].

A=RREQ sent
B=RREP sent
C=RERR sent
D=DATA sent and IP source address is not of monitored node
E=RREQ received
F=RREP received
G=RERR received
H=DATA received and IP destination address is not of monitored mode

The following mapping is from [SaB03]:

A protocol trace may consist of the following sequence

l_1 = (EAFBHHEDEBHDHDHHDHD,…)

A set of genes used for pattern matching is also defined to develop the antigen.

Using the following list:

Gene1=#E

Gene2=#(E*(A or B))

Gene3=#H

Gene4=#(H*D)

l_1 can be mapped to the antigen l_2 = (3 2 7 6)

To facilitate bit matching the l_2 antigen is encoded to a string of ones and zeros where the value of the gene is represented by a one in the n^{th} bit. For example, l_2 would map to the following:

l_3 = (0000001000 0000000100 0010000000 0001000000)

l_3 is the final representation of a single antigen. Antibodies have the same representation except they can have multiple ones in each gene string. If the antibody has a one in every position that an antigen has a one, it is considered a match.

For example, the antibody:

a_1 = (1100001001 1000010110 0011001000 1001000100)

would match antigen l_3 because it has a one in every position that l_3 does.

To prevent a false positive for simply matching an antigen to an antibody, a threshold equation is used to ensure that more than one detector matches the same misbehaving node. The equation for the threshold detection is shown in (4.1) [SaB05].

$$\frac{M_n}{n} > \theta_{\max}\left(1+\frac{\xi(\alpha)}{\sqrt{n}}\sqrt{\frac{1-\theta_{\max}}{\theta_{\max}}}\right) \tag{4.1}$$

where M_n is the number of detectors that detected the node, n is the number of detectors that monitor the node, $\theta_{\max}$ is the maximum bound for false positive detection, and

$\xi(\alpha)$ is the (1- α)-quantile of the normal distribution. If the equation evaluates true, the node is considered malicious.

4.5.3.3 Design of MOAIS. To design the MOAIS, the symbolic notation of the problem is developed into a working algorithm. A representation of the operation of the algorithm is shown in Figure 4.27. From this depiction, it is shown that a set of antibodies is randomly generated and then trained using both positive and negative selection to arrive at a useable set of detectors. It is important to note that this training must be done within a trusted environment. This is analogous to the HIS creating antibodies in the bone marrow of the human body.

Once the detectors are developed, they are able to detect both suspicious and malicious nodes in the network. The way the IDS differentiates between the two is a malicious node must be detected as suspicious by a threshold number of other nodes. If it is below the specified threshold, it is labeled as only a suspicious node and no further action is taken.

Should a node be labeled as malicious, the IDS takes appropriate action and then runs through a clonal selection process which allows the IDS to increase the number of detectors that found the malicious node. These new detectors are created from a copy of the successful detector and mutated to create small variations. These mutations are run through a negative selection process to be sure that they do not detect non-malicious nodes (self). The newly created detectors replace detectors with low fitness values.

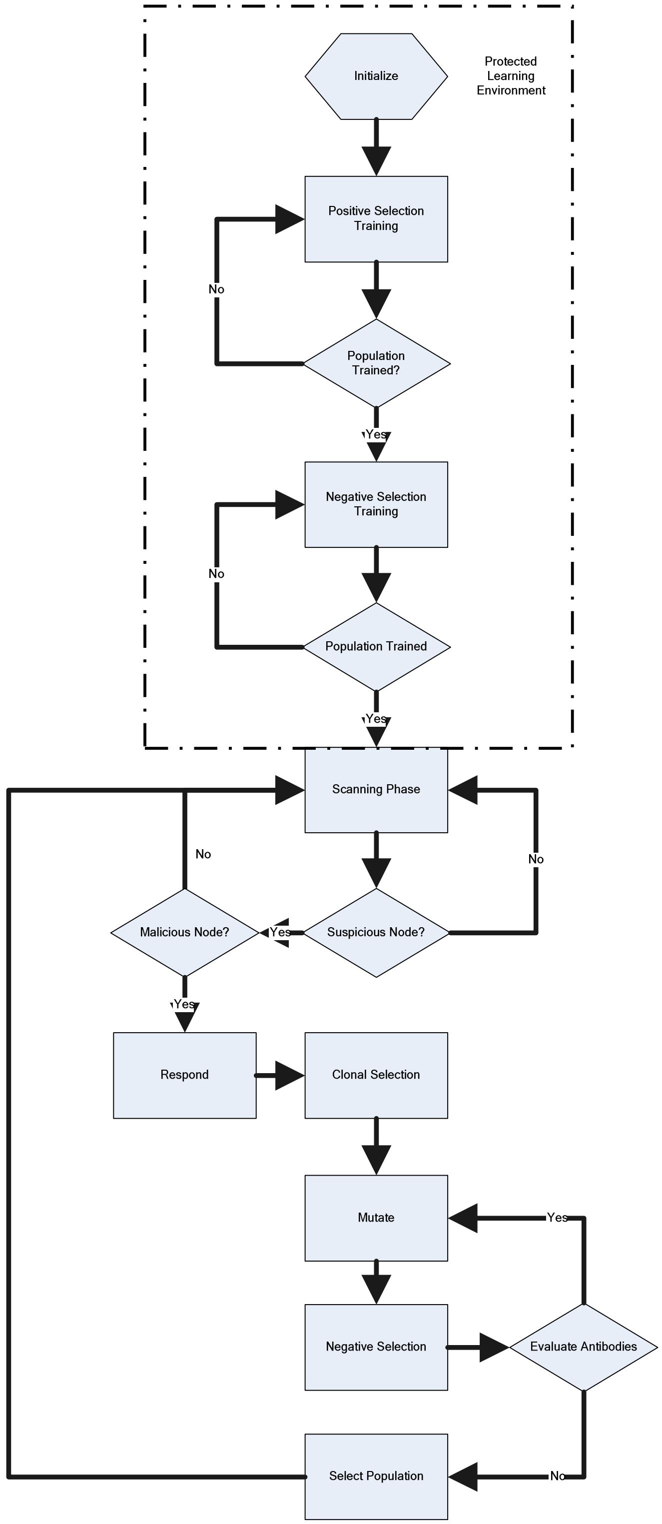

Figure 4.27: Operation of MOAIS for IDS of MANET.

As stated earlier, the algorithm is a multi-objective one with the following fitness functions based on effectiveness and efficiency

$$F = w_1(f_{effectiveness}) + w_2(f_{efficiency}) \tag{4.2}$$

$$\text{where } w_1 + w_2 = 1 \tag{4.3}$$

where $f_{effectiveness}$ is measured in terms of the false positive rate and $f_{efficiency}$ is measured as the time until classification.

To attain the desired performance metrics from the IDS, many parameters have to be determined. A partial list of tuning parameters includes learning time in the protected environment, number of antibodies, the size of the antibodies, false positive threshold, size of duplication in clonal selection, and rate of mutation [SaB03].

4.5.3.4 Testing and Evaluation. The design of the MOAIS is tested using a simulation environment for MANETs called Glomosim [SaB03]. By varying the targeted false positive classification rate, the authors produce plots of false positive effectiveness ratios (effectiveness) and time until classification (efficiency). The plots are combined to produce the Pareto plot shown in Figure 4.28. By minimizing the false positive classification rate, the results move down to the right of the line favoring effectiveness over efficiency.

The results also show that the clonal selection function of the algorithm has significant effects to response time [SaB03]. While decreasing the detection time, clonal selection also reduces the false positive rate. The rational explaining this is if a node was exposed to a malicious node in the past, it will be easier to recognize another malicious node in the future by using clonal selection [SaB03].

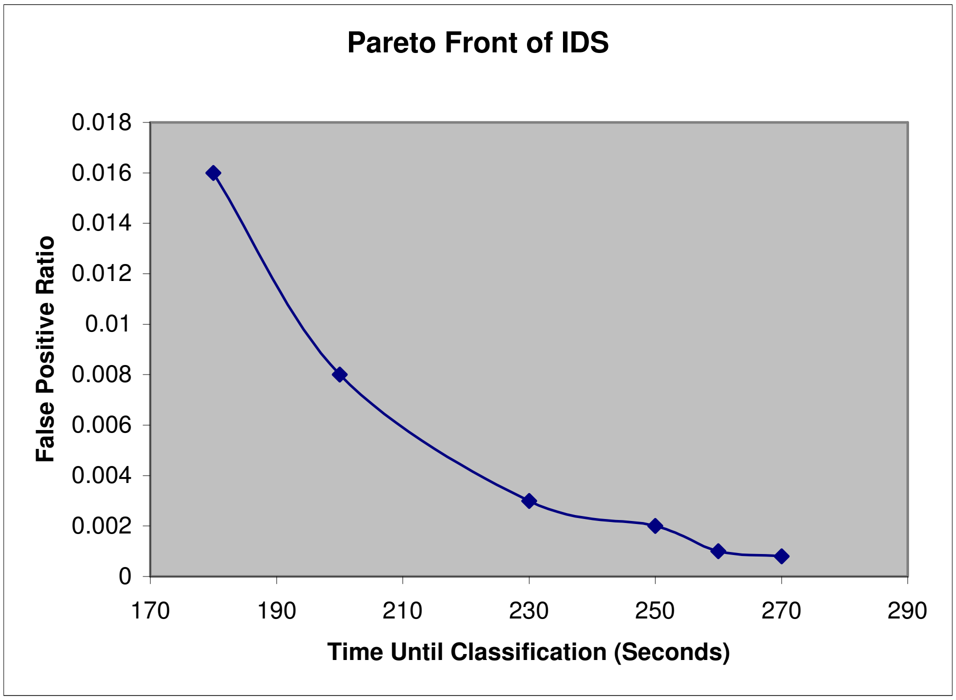

Figure 4.28: Pareto Plot of IDS Results.

With the Pareto plot, a decision-maker can objectively make decisions about the architecture of the IDS. Depending on the application, a low false positive rate might be required. In a different network, the time until classification is more important. The Pareto plot allows the decision-maker to make informed trade offs with the knowledge of how such trade offs will affect the parameters of the IDS. If the Pareto plot shows unacceptable IDS behavior in all variables then the IDS may have to be redesigned to shift the plot to the left for a more acceptable response.

4.5.3.5 Summary of MOAIS Research in MANETS. Although there are many successful implementations of IDSs in wired networks, due to the inherent differences between wired and wireless networks, these solutions fail for a wireless network. This research explores the issue of using a MOAIS to solve this problem. Because a wireless

ad-hoc network is structurally similar to cells roaming around in the human body, an artificial immune system type approach appears to be very promising in detecting malicious nodes. The AIS allows the IDS to be lightweight yet effective which is paramount within the limited capabilities of the devices that are typically in a wireless network.

To clearly understand the problem, this research defines the intrusion detection problem symbolically which eases the transition from the problem domain to the algorithmic domain. The results of the current implementation of the MOAIS are also discussed.

4.5.4 A Retrovirus Inspired Algorithm for Virus Detection (REALGO) As part of researching vulnerabilities to construct attack trees, research is conducted that deals with building a virus scanner that can detect viruses that have undergone metamorphism. In the search for a robust and efficient algorithm to be used for computer virus detection, an AIS genetic algorithm is developed based on the human immune system's use of reverse transcription Ribonucleic acid (RNA). The REALGO algorithm provides a memory such that during a complex search the algorithm can revert back to and attempt to mutate in a different "direction" to escape local minima. In lieu of non-existing virus generic templates, validation is addressed by using an appropriate variety of function optimizations. It is empirically shown that the REALGO algorithm finds "better", or statistically superior, solutions than other evolutionary strategies in four out of eight test functions and finds equally "good", or statistically equal, solutions in the remaining four optimization problems.

4.5.4.1 Motivation for Virus Scanning Algorithm. Contemporary computer systems are integral in daily life and can accomplish increasing complex tasks. With this complexity, however, come system vulnerabilities. There are so many interdependencies and relationships between entities in a computer system that a malicious user does not need to find a vulnerability to attack, rather they must choose which one to attack.

In 1995, the first year the Computer Emergency Response Team (CERT) Coordination center published statistics for vulnerabilities, they reported 171 different vulnerabilities. In 2004, the number has increased to 3,780 [Cer05]. In 1995, there were 2,412 security incidents reported. The number has increased to 137,529 in 2003 [Cer05]. Viruses exploit these vulnerabilities to infect their hosts. CERT has stopped counting separate incidents in search for a more meaningful statistic of attacks since it is so easy to attack a computer system. Of course, other computer security issues with increasing concern include Trojan horses, bots, backdoors, dialers, worms, adware, and spyware. The proposed AIS system can address these threats as well as viruses.

Currently, most virus protection for computers is implemented using signature recognition derived after analyzing known viruses [Szo05]. Although this method has been successful, a time is rapidly approaching when such methods will be inadequate. As viruses are constantly mutated and tweaked to avoid detection, the signature list becomes larger, possibly approaching seven figures. Another problem is that viruses are only detected after they have been initially discovered, analyzed, and signatures have been distributed. This process can be very slow and wastes precious time while a fast replicating virus quickly renders a networked computing system useless.

What is needed is a system that can combine known information from past viruses with a type of prediction for future viruses. Utilizing a computing system based on an artificial immune system may be able to achieve this. If a computer can be "immunized" initially, "learn" from viruses and other computers, and use that knowledge with evolutionary algorithms to come up with new signatures for probable viruses, viruses are effectively eliminated before they have time to replicate. A variety of AIS operators have been suggested for this purpose [CaT02, Das02]. This research proposes a new type of AIS to detect computer viruses based on the REtrovirus ALGOrithm (REALGO). REALGO is based on the concept of reverse transcription RNA (Figure 2.16) as found in biological systems [Kle03]; i.e., Reverse Transcription Ribonucleic Acid (RT-RNA).

The REALGO uses a random antibody initiation process and then, using known virus signatures (antigens), "trains" these antibodies through a genetic algorithm (GA). A distance measure is usually a fitness function representing the distance (Euclidian, Hamming, …) between antigen bit-string and the antibody bit string. After an initial learning phase, the antibodies are released into the computer system to recognize virus patterns. The antibodies constantly evolve using mutation (affinity maturation) to identify new viruses, adapting and diversifying the antibody memory. When a new virus is detected in a trusted network a new retrovirus signature is created and sent to other network systems to "immunize" them. To keep the number of antibodies to a minimum and increase performance, a temporal aspect is employed. Basically, antibodies that have not been used die after a specified time period (metadynamics) [Gar05, TiK04]. If an antibody is deemed particularly important, it can be protected from elimination using a

protection flag. On the other hand, antibodies that have been activated previously can trigger a secondary response (Figure 4.29), as is done in the biological immune system. Thus, in response to a repeated attack, the memory provides for increased efficiency (identification speed) and effectiveness.

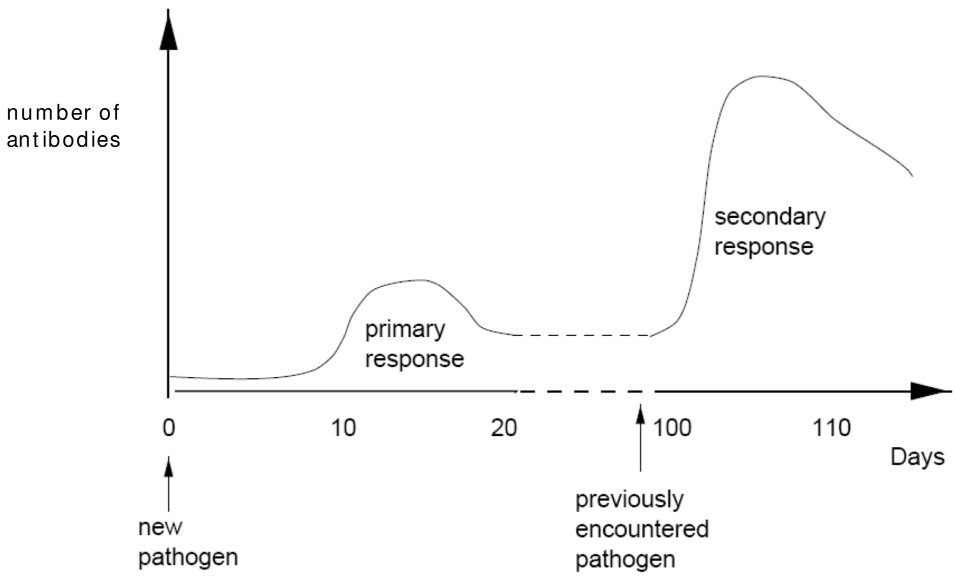

Figure 4.29: Antibody Primary and Secondary Response [Hof99].

An important concept of antibody development for detecting virus attacks is the knowledge extraction or learning phase. This phase is further decomposed into two distinct parts; training for positive selection and negative selection. Positive selection is the ability of the antibody to detect a virus. Negative selection ensures that the antibody does not trigger on the computer's concept of *self*. It is undesirable for the antibody to identify a normal process or file (self) as a virus, in other words, a false positive. The antibodies can be trained for positive selection by using a genetic algorithm on the initial population of randomly generated antibodies using a comparison operator against known

virus signatures. This effectively trains the antibodies to detect known viruses though these antibody-virus signature pairs are probably not exact matches. Instead they match within a specified threshold [HaL00, TiK04] given a distance metric. Because they are not exact matches, the newly trained antibodies must be trained against detecting self using negative selection techniques.

To filter the antibodies, a fitness function matches antibodies against known good processes and programs in the computer (self). It is critical that the computer system is not already compromised by a virus since it would filter out the specific virus detection capability. If an antibody matches a good process string or program string within a certain threshold, it must be discarded. To ensure that the final population is of the desired size, enough antibodies in the positive selection phase must be generated so some can be discarded. If the number of antibodies drops below the desired population during negative selection training, new antibodies are randomly generated and trained using positive selection. If, after negative selection training, there are extra antibodies; the ones with the lowest positive selection fitness values are discarded.

The REALGO AIS as designed incorporates all of the salient features mentioned at the beginning of this section. Similar systems for detecting computer attacks have been developed and implemented by Forrest, et al. in [FoH00, HoF00], Harmar, et al. in [HaL00, HaW02], and Williams, et al. [WiA01]. The main difference is the use of RNA as a *memory* structure which the search can use to revert to a previous "good" solution if the search stagnates and fails to produce a "better" solution. After reverting back to this "good" solution, the search can proceed in a different "direction" to find a "better"

solution. This operation is expected to increase the performance of the search algorithm in both efficiency and effectiveness.

4.5.4.2 A Symbolic REALGO Model To formally understand and apply the REALGO algorithm, a symbolic notation is provided for a detailed presentation of the algorithm. A flowchart is also provided in Figure 4.30. The following symbolic model is based on notation for genetic algorithms in [Bac96]:

The time variable, *t*, is initialized and the population of antibodies is generated randomly,

Initialize: $t = 0$ and $P(0) = \{\overrightarrow{a_1}(0), ..., \overrightarrow{a_u}(0)\}$ where $P(0)$ is the initial population set of antibodies and $\vec{a} = (a_1, ..., a_l) \in \mathbb{B}^l$ which consists of a bit string of length *l*.

Next, the population of antibodies is evaluated against the training antigens (known virus signatures). For the first iteration this evaluation is equal to zero, as antigens have not yet been imported.

Evaluate: $f(\overrightarrow{x_1}(0), ..., \overrightarrow{x_u}(0))$; Evaluate population of antigens using fitness function, *f*.

while not terminate (P (t)) do; The termination condition is based on a threshold of matching the antibodies to the antigens. If not terminating, the RNA structure is imported which consists of antigens. In this application, the antigens are a string of virus signatures, *V(t)*. For the first iteration, they are known virus signatures from previous virus knowledge bases.

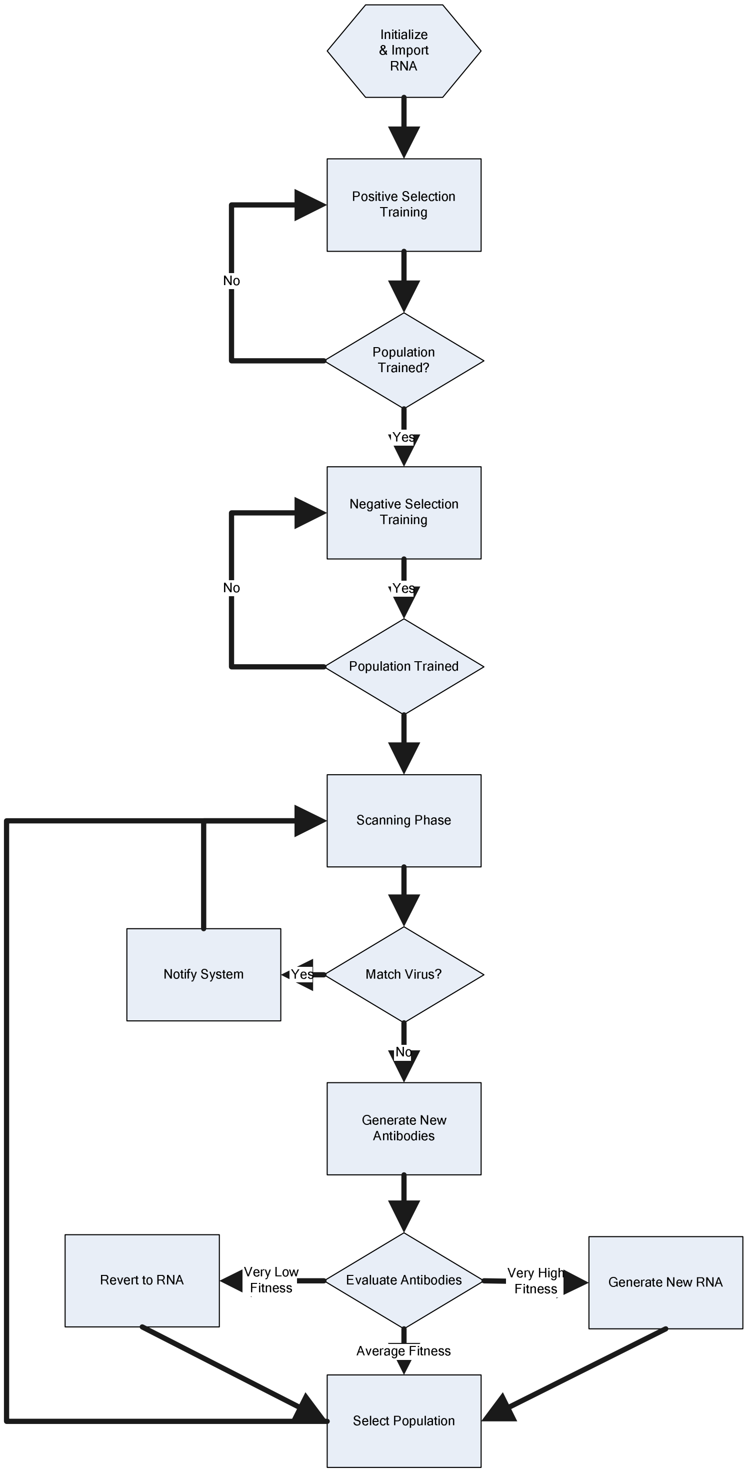

Figure 4.30: Flowchart of REALGO Algorithm.

Import RNA: $V(t) = \{\overrightarrow{v_1}(t), ..., \overrightarrow{v_u}(t)\}$ where $\vec{v} = (v_1, ..., v_l) \in \mathbb{B}^l$ which consists of a bit string of length *l*.

After importation, the population is compared with the imported RNA and the population member is replaced with the RNA with probability p_1, which is empirically chosen, if the fitness of the RNA is higher. A protection variable, ρ, is checked that allows the algorithm to protect population members. This prevents the replacement of members that have previously had a high probability of matching a known virus.

Select RNA: $\vec{a}_k(t) = \vec{v}_j(t)$ with probability p_1 if $f(\overrightarrow{v_j}(t)) > f(\vec{a}_k(t)) \forall j \in \{1,..,n\}$ and $\rho_k \neq 1$ (Cell is not protected);

To create new antibodies mutate members of the population to create λ children $\vec{a}''_k(t) = m'_{\{p_m\}}(\vec{a}'_k(t)) \forall k \in \{1,...,\lambda\}$ where m is a mutation operator that flips each bit with an empirically derived probability p_m.

As part of the process of selecting the next generation of antibodies, evaluate the children just produced.

Evaluate: $P''(t) = f(\{\overrightarrow{a_1}''(t), ..., \overrightarrow{a_\lambda}(t)\} : \{\overrightarrow{x_1}''(t), ..., \overrightarrow{x_\lambda}(t)\})$

Now, check to see if the antibodies fail to match any antigens by a specified threshold. If so, then further check if the string evolved from a RNA strand. If it did, revert back to the RNA string (memory) with probability p_2. The purpose of this check is to revert back to a known good starting point if the search for a good antibody stagnates. This operation allows the algorithm to explore from a known good starting point but in a different direction than one that produced the stagnant solution.

If $\exists f(\vec{x}''_k(t)) < \varepsilon_1$ then (If fitness falls below threshold ε_1)

If $\xi_k = 1$ (RNA based cell indicator) $\vec{a}_k(t) = \vec{v}_k(t)$ with probability p_2 (Revert back to RNA)

To exploit an area of good solutions, the algorithm spawns new strands of RNA if a particular antibody evaluates very high based on a set threshold. If it does, the algorithm creates *n* copies to be used in developing the next generation.

If $\exists f(\vec{x}''_k(t)) > \varepsilon_2$ then (If fitness is above threshold value ε_2)

$V(t) = \{\overrightarrow{v_1}(t), ..., \overrightarrow{v_n}(t)\}$ (Create *n* copies of RNA)

The next generation is now determined using $\mu + \lambda$ selection by selecting the best μ individuals out of the $\mu + \lambda$ population [Esh00].

Selection: $P(t+1) = s_{(u+\lambda)} P''(t)$ where *s* is the selection operator

The time variable is iterated and the algorithm loops back to the evaluation of the children until termination $t = t + 1$.

Now that positive selection is completed, the algorithm must check the antibodies against the system's concept of self to prevent the system from attacking good processes and programs. It is important that the system is in a known "clean" configuration at this time. If this is not true, then any viruses in the system are identified as self and the algorithm will actually protect them. This concept of training the antibodies against the concept of self is known as *negative selection*.

If $\exists f(\vec{x}_k(t)) > \varepsilon_3$

then $P(t) = P(t) \setminus \{x_k(t)\}$

If $P(t)$ falls below a specified number of elements, new ones are generated using the positive selection part of the algorithm followed by negative selection until the specified number of antibodies is achieved.

Now that training is complete, the algorithm is ready to release the antibodies into the network computer system to patrol for viruses. As programs and/or processes are started, copied, or accessed in any way, the antivirus scanner compares them to the antibodies in the system. If they match within a specified threshold, the antivirus program checks to see if the antibody is *mature* or *immature*. If immature, the program needs a secondary confirmation to take action on the suspected virus. This confirmation comes from the user. Once a virus is confirmed, the antibody is considered to be mature, sets a flag on the antibody data structure, and does not require confirmation for future viruses. Imported signatures can be flagged as mature and thus not require confirmation. This is analogous to a *secondary response* in the human immune system [FoH00].

Although not implemented in this system, once a virus antibody is confirmed, it can be uploaded to a central repository where it is further checked by anti virus researchers. If it qualifies, the antibody is sent to other users to vaccinate them on the new virus strain. The advantage of this process is after the first variant of a virus discovered by *any* user, the inoculation is sent out within minutes to every other user which effectively stops the propagation of the virus almost immediately. This concept is discussed in [LaM99].

This system does not require the researchers to actively find the virus. Instead, the virus antibody is sent to them automatically. This would drastically reduce the

amount of time it takes to detect new variants of viruses by using a distributed detection system.

To search for antibodies that match new virus variants, the REALGO program uses a genetic algorithm with mutation to generate new antibodies. If known good building blocks exist in any of the antibodies, they can be protected and remain intact throughout the operations.

When selecting the next generation based on a fitness value, a choice is made between the children generated by the REALGO operations, the parents, and the antibodies received by other systems. The fitness value also takes into account the age of the antibody and the number of activations it (or its building blocks) has encountered. If there is an "old" antibody that has rarely been used, it might be deleted and a replacement is generated based on the current threats. This allows the program to remain agile by maintaining a smaller "vaccine" file. It is also assumed that at least one system finds the virus and passes the "vaccine" to the rest of the system. This leverages the distributed nature of the interconnected computer systems to spread the burden of maintaining a large signature file.

4.5.4.3 Evolutionary Computation. A genetic algorithm is the natural choice for both training the antibodies and modifying antibodies in the system once they are deployed because of the binary allele's genotype structure. A generic psuedocode for the complete anti virus scanner follows:

REALGO Psuedocode

```
1:    Initialize population of antibodies (random)
2:    Introduce known antigens (virus signatures)
```

```
for fitness of antibodies not sufficient do
    Compare antibodies to antigens (Genetic Algorithm)
    Evaluate antigens based on bit matching to antibodies
    Select x number with highest fitness
    Generate children via mutation
end for
if antigens based on bit matching match known good programs/processes then
discard
while (true) do
    if programs/processes match antibodies beyond specified threshold then
        generate warning
        Update data in antibody reflecting number of matches (fitness)
        if virus is confirmed then
            send a copy of antibody to other systems
            clean system
    Generate new children by mutation
    Select parents based on fitness
    if fitness is high (ε*average fitness) then
        generate n copies of DNA to become RNA with prob p_1
    if fitness stagnates for g generations and RNA exists then
        revert back to RNA with prob p_2
    Select population based on fitness and age
end while
```

The natural data structures for the virus signatures are binary strings shown in Figure 4.31. The strings are encoded with the signature data as well as a mature flag, a protection flag, an RNA exists flag, and a pointer to any associated RNA structure that the string initially evolved from. RNA structures consist of the same types of strings which means that nesting of the RNA structures can occur.

S	i	g	.	.	.	.	.	.	.	M F	P F	R E	→

Figure 4.31: Generic Signature Data Structure.

For mutation, a Cauchy distribution is used as this has been shown to have the ability to make long jumps to escape local minima as compared to a Gaussian distribution

[YaL97]. A simulated annealing time function also increases exploitation after an initial period of large exploration. The combination of these two operators makes the search *memetic*, otherwise known as a combination of global and local search techniques.

To calculate the fitness value, a matching rule is used [FoH00]. If the bits of the antigen and the antibody match for r-contiguous bits then the fitness value is computed to be r. The range of r is $0 \leq r \leq l$ where $l =$ signature string length

Experimental data is used to determine the ε multiplicative values for generating RNA and the number of generations of stagnation (represented as the percentage, g, of total number of generations in search) before reverting back to the memory RNA structure. These are basically tuning parameters that determine the rates of exploration and exploitation. It is assumed that the user has a general idea of the problem domain *landscape* and is able to initially set these parameters based on the expected search landscape.

4.5.4.4 Design of Experiments. To evaluate the REALGO AIS algorithm, various procedures are defined based in part on the assessment criteria of Garrett [Gar05]. The first step in testing the system is to ensure that the REALGO algorithm produces good results in complex search landscapes. One could consider that a computer virus search landscape would probably consist of, or be close to, "needles in a haystack" or delta functions on a discrete grid. But a realistic generalized virus landscape is not available since there are no known complex models of virus selection benchmarks. Thus, other evaluation benchmarks need to be defined to evaluate the new REALGO algorithm. Function optimization replaces virus benchmarks for algorithm performance analysis.

The REALGO algorithm is coded in C++ and is tested against eight well known test functions that represent a variety of desired search landscape characteristics as shown in Table 4.5 [YaL97]. The initial research is limited to these eight test functions as a proof of concept for the algorithm. Once the desired operation of the algorithm is confirmed based upon metrics, it is integrated into the detection system.

Table 4.5: Test Functions Used in Testing REALGO [YaL97].

Test function	S	f_{min}
$f_1(x) = \sum_{i=1}^{n} x_i^2$	$[-100,100]^n$	0
$f_2(x) = \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$	$[-10,10]^n$	0
$f_3(x) = \sum_{i=1}^{n} (\sum_{j=1}^{n} x_j)^2$	$[-100,100]^n$	0
$f_4(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	$[-30,30]^n$	0
$f_5(x) = \sum_{i=1}^{n} (\lfloor x_i + 0.5 \rfloor)^2$	$[-100,100]^n$	0
$f_6(x) = \sum_{i=1}^{n} (-x_i \sin(\sqrt{\lvert x_i \rvert}))$	$[-500,500]^n$	-12569.5
$f_7(x) = \sum_{i=1}^{n} [x_i^2 - \cos(2\pi x_i) + 10)]$	$[-5.12,5.12]^n$	0
$f_8(x) = \frac{1}{4000} \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}}) + 1$	$[-600,600]^n$	0

Employing this functional optimization approach, results are averaged over 50 runs and compared against those in [YaL97]. Tuning parameters are developed empirically using limited knowledge of the landscape of each problem. For example, the f6 function is graphed in a single dimension (Figure 4.32) to understand the problem domain landscape. It is easy to see that there is a local minimum quite a distance away from the global minimum. This "egg carton" landscape could be thought of as a set of

delta functions on a numerical grid possibly reflecting a virus signature landscape. To be able to jump far in this landscape, the simulated annealing time constant *t* is adjusted at a slower rate than a function such as f1 (Figure 4.33) which requires a much finer level of resolution in its search.

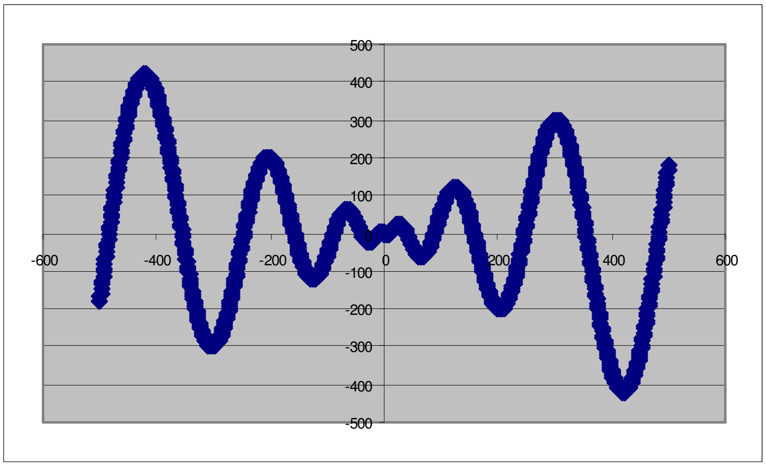

Figure 4.32: The Graph of f6 in One Dimension. This data is used to estimate the simulated annealing constant for mutation when the search is expanded to 30 dimensions.

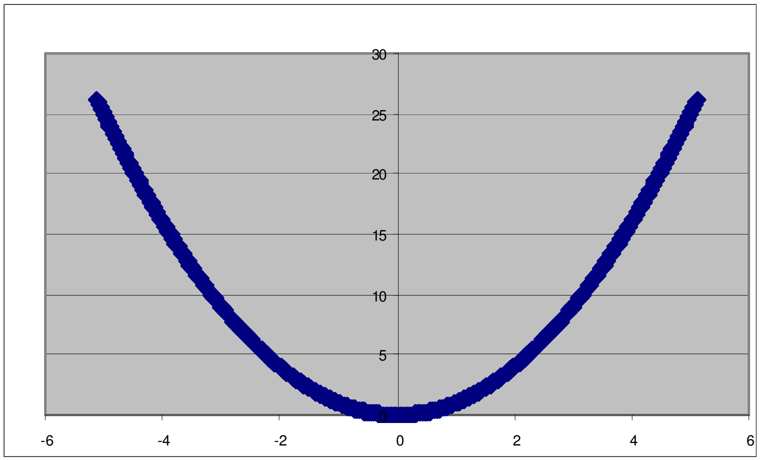

Figure 4.33: Graph of f1 in One Dimension. This function is much smoother and requires a higher resolution search thus a smaller mutation operator which is controlled by the simulated annealing constant *t*.

Once libraries of signatures and viruses are obtained, testing begins with a small sample to ensure the algorithm functions as desired in this functional optimization domain. This small sample contains 100 antibodies. After initial validation, system testing begins using the complete virus binary library. Five-fold cross validation is used to measure detection and false positive rates [Koh95]. The size of the libraries obtained determines the population size and the number of virus binaries tested. Based on similar work from [ScE01], a binary library of 3000+ is assumed to be a reasonable size for comparison purposes.

REALGO system testing is executed on a single computer platform with a 2.53 GHz Pentium 4 processor and 512 MB of random access memory (RAM) running the Windows XP Professional operating system. Note that virus detection measurement methods and rates are extremely non-standard [GoF96]. Once data is available from the REALGO experiments, it is compared against other published virus detection algorithms such as those in [ScE01]. The false positive rate should be close to zero for all non-virus binaries tested. Future work should include testing the artificial immune system on a distributed network. As more signatures are involved in a distributed network, the detection rate should increase. This would demonstrate the benefit of using a network of computers for REALGO virus detection. With the computers on the network cooperating, the new viruses have little chance to propagate.

4.5.4.5 Experimental Results. The REALGO algorithm is tested against eight benchmark functions as a proof of concept for the algorithm. The associated results of the REALGO algorithm are shown in Table 4.6. These results are compared to those

from [YaL97] (Table 4.7) with the resulting t-tests shown in Table 4.8.

Table 4.6: Results of REALGO Algorithm on Eight Test Functions.

	Number of Generations	RNA Mean Fitness	RNA Standard Deviation	No RNA Mean Fitness	No RNA Standard Deviation
f1	750	9.44e-5	1.42e-5	1.63e-4	2.02e-5
f2	1000	3.37e-2	2.75e-3	4.44e-2	2.99e-3
f3	2500	3.54e-1	7.55e-2	4.41e-1	6.77e-2
f4	7500	3.27	4.06	3.14	4.09
f5	750	0	0	0	0
f6	4500	-12563.3	5.53e-1	-12577.3	109.689
f7	2500	9.44e-5	1.42e-5	6.16e-3	7.16e-4
f8	1000	1.52e-2	2.42e-3	2.46e-2	2.64e-3

Table 4.7: Results of FES and CES on Eight Test Functions [YaL97].

	FES Mean Fitness	FES Standard Deviation	CES Mean Fitness	CES Standard Deviation
f1	2.5e-4	6.8e-5	3.4e-5	8.6e-6
f2	6.0e-2	9.6e-3	2.1e-2	2.2e-3
f3	1.4e-3	5.3e-4	1.3e-4	8.5e-5
f4	33.28	43.13	6.69	14.45
f5	0	0	411.16	695.35
f6	-12556.4	32.53	-7549.9	631.39
f7	0.16	0.33	70.82	21.49
f8	3.7e-2	5.0e-2	0.38	0.77

Using the student's t-test to compare the results, the REALGO algorithm produces better results when the RNA function is enabled on four out of the eight functions. On the remaining functions, it produces equally good results that are not statistically different whether or not the RNA function is enabled. Compared to the FES (Fast Evolution Strategies) algorithm from [YaL97], REALGO outperforms FES on three of the eight functions and produces statistically equivalent results on the remaining five.

Compared to CES (Classic Evolution Strategies), REALGO outperforms on two out of the eight functions and is shown to be statistically not different on three other functions. Two of the three functions that CES performs a better search (f1 and f2) are the same ones that CES outperforms FES on in [YaL97]. CES performs better on these searches due to the simpler landscapes and the fact that CES is not burdened by the additional overhead that FES and REALGO have. On the more complex searches, the overhead is needed and is reflected in the improved results.

Table 4.8: Student t-tests for REALGO Versus Other Algorithms. (REALGO versus RNA Memory Disabled, REALGO versus FES and CES Results). * Indicates Statistical Significance.

	RNA-No RNA	RNA-FES	RNA-CES
f1	2.47e-3*	1.79e-2*	3.25e-3*
f2	1.63e-4*	2.18e-2*	6.26e-4*
f3	2.13e-1	1.26e-3*	1.25e-3*
f4	9.71e-1	3.15e-1	6.26e-1
f5	1	1	4.13e-1
f6	2.68e-1	7.44e-1	5.24e-3*
f7	4.43e-3*	4.90e-1	2.93e-2*
f8	1.81e-4*	5.11e-1	4.97e-1

To graphically illustrate the results of REALGO, the means of 50 runs with a one standard deviation error bar are plotted. The other test function plots reflect similar pictorial results. The results for f1 and f8 are shown in Figures 4.34 and 4.35 respectively.

Through empirical testing, design parameters are determined which generate the best results based on the associated fitness functions. These values are shown in Table 4.9.

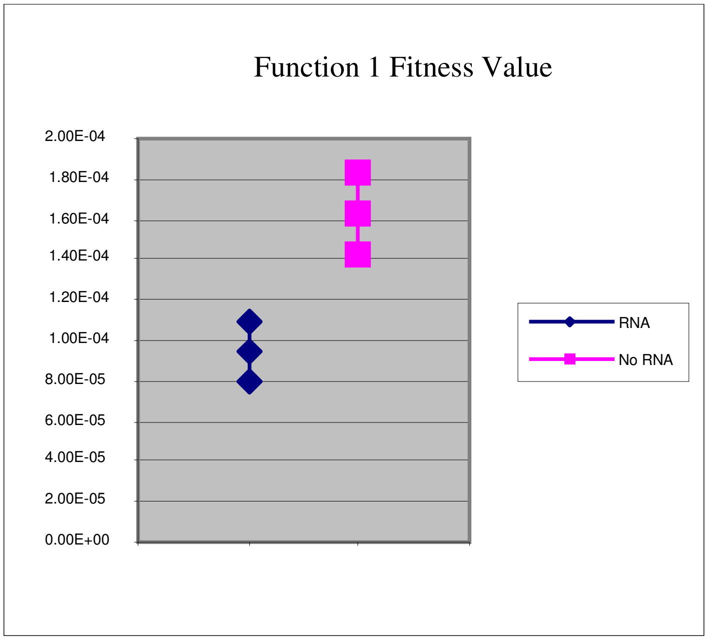

Figure 4.34: Results of REALGO on function 1. The results with RNA enabled are statistically better.

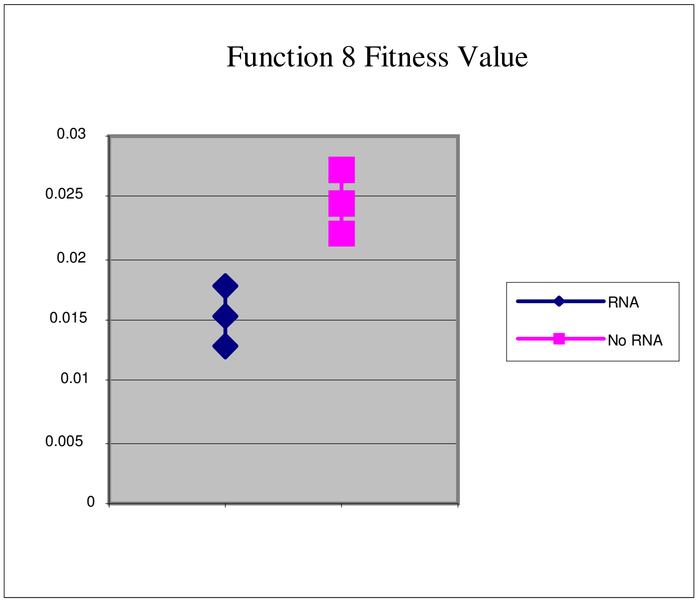

Figure 4.35: Results of REALGO on function 8. The results with RNA enabled are statistically better.

Table 4.9: Empirically Derived Tuning Parameters for REALGO Algorithm.

Func	f1	f2	f3	f4	f5	f6	f7	f8
ε	1.03	1.03	1.03	1.03	1.03	1.03	1.03	1.03
Prob p_1	20	20	20	20	20	20	20	20
g	10%	10%	10%	10%	10%	10%	10%	10%
Prob p_2	5	5	5	5	5	5	5	5
t	500	500	0.1	500	500	0.1	500	0.1

4.5.4.6 Summary of REALGO Research. The results show that the REALGO algorithm is superior for optimizing complex functions but not necessarily for easier ones, because the REALGO algorithm adds complexity to the search that is not needed for simple searches. Once the complexity of the search landscape is greater than that of the algorithm, the REALGO algorithm is superior. For a simple search, something as basic as a hill climber search technique suffices. If the complexity of the REALGO algorithm is used with the simple landscape, the efficiency is reduced because the complexity is not required to find the solution. Conversely, with a complex function such as f6 (Figure 4.32), a simple hill climber search technique usually fails to find the best solution resulting in very poor effectiveness.

Preliminary results show that the REALGO algorithm does indeed provide a superior search for complex landscapes due to its ability to revert back to a previous good solution if the search stagnates. Rather than resetting to a new starting point, a search in a new direction from this previous good solution is attempted without having to waste generations for the initial convergence. The next step is to integrate it into a complete

virus detector.

This research proposes a novel artificial immune system to detect computer viruses. The use of reverse transcription RNA as a *memory* structure for a global search has not been proposed or implemented in any literature. The use of the reverse transcription RNA can be generalized to other global search genetic algorithms to increase their performance. Additional applications of the REALGO AIS approached could include multiple robot/UAV control, fault diagnosis, and network intrusion detection (IDS). The REALGO algorithm can be modified from virus detection to a network IDS by changing the determination of self and non-self to represent network characteristics rather than local application characteristics.

4.6 Summary of Applications

This chapter presents applications that are analyzed using the attack and protection tree methodology. Each of the applications adds a new element to the framework as different characteristics of their respective systems are discovered when implementing the security analysis. The application domains used include a computer network, a DHS information sharing network, an online banking system, and a MANET.

Additional research on specific attack and protection leaf nodes is also presented. This includes using virus protection as software protections, specifically metamorphism. The virus protections offered can be used as specific protections in a computer software application attack scenario. The implementation of a multi-objective artificial immune system as an intrusion detection system for a MANET is also discussed. The MANET multi-objective IDS describes the operation of a possible algorithm that can be used in a

future IDS. Finally, the development of a novel algorithm to detect metamorphic viruses is presented and tested. The REALGO algorithm research can be used as a detection tool for computer viruses that would also be part of a protection scheme. This could also be used as a detection mechanism as part of a larger protection scheme.

V. Validation of Methodology Using the Schematic Protection Model

To validate the methodology presented in this research, it must be shown that the use of protection trees results in a more secure system. A secure system in this context is defined as a system that does not allow unauthorized users' to obtain a specific authorization that the protector is trying to defend. To accomplish this, the construct of protection trees is mapped to the Schematic Protection Model [San88]. The key idea in SPM is the notion of the protection type [Bis03]. The protection type of an entity determines how control rights affect the entity. The protection type of an entity is determined when the entity is created and cannot be changed. SPM consists of entities, tickets, links, and filters. Although the actual mapping is presented later, for explanation purposes a notional mapping is presented below.

5.1 An Introduction to the Schematic Protection Model

Entities in the SPM can be mapped to subjects and objects in protection trees. A subject may be a computer user or process while an object could be a file. A ticket describes a right held over an entity. A ticket in a computer system could be *read*, *write, execute, copy, etc.* A link describes the relationship between entities. A filter establishes conditions that allow tickets to be transferred. In a protection tree, the filters are the protections.

The value of mapping protection trees to the SPM is it can be used to show, given certain assumptions, the protection tree results in a secure system. If it is theoretically

shown that the system is secure then the only reason an actual system would not be secure is due to its implementation being different than the model.

5.2 Implementing the SPM on the Online Banking Application

To demonstrate the safety analysis of attack and protection trees, the online banking application from Chapter IV is used since it consists of networked computers with clear adversaries and protectors. The motives of the attackers, costs for protection, and amount of losses are also metrics that are easily measurable.

5.2.1 Mapping Online Banking Application to SPM. Consider a simple online banking system with users, merchants, criminals, and bank employees. Other subjects include the bank server and a transfer network. Objects in the system are account numbers, account identification, account passwords, debit card numbers, debit card personal identification numbers (PIN), account balances, money, bank employee identification, and bank employee passwords. A complete list of the entities and their associated abbreviations are listed below.

<u>Subjects:</u>

Any Subject (TS)
User (U)
Merchant (M)
Criminal (C)
Transfer Network (TN)
Bank Employee (BE)
Bank Server (BS)

<u>Objects:</u>

Any Object (TO)
Account Number (AN)
Account ID (AID)
Account Password (AP)
Debit Card Number (DCN)
Debit Card PIN (DCP)
Account Balance (AB)
Money ($)
Bank Employee ID (BEID)
Bank Employee Password (BEP)

5.2.2 Initial State of System. With the entities defined, the initial state of the system is determined. The tickets each object possesses in their domains are listed

below. Rights are defined as r = read, w = write, g = grant, t = take, x = execute, and c = copy.

dom(U) = {AN/rc, AID/rc, AP/rwc, DCN/rc, DCP/rc, AB/rc, $/rc, TS/g}; Explaining the notation, user U can: read his account number, read his account ID, read or write his password, read his debit card number, read his debit card PIN, read his account balance, and read (access) the money in his account. He may also copy any of these rights (copy flag associated with every right) to any subject he chooses through the grant right.

dom(M) = {TN/g}; the merchant can initially grant any rights it possesses to the transfer merchant.

dom(C) = {AID/t, AP/t, BS/t, DCN/t, DCP/t, TN/x}; The criminal can take an account ID, password, any right from the Bank Server, the debit card number, debit card PIN, and has execute rights (access rights) on the transfer network.

dom(TN) = {DCN/rc, DCP/rc}; the transfer network can read and transfer (copy) the debit card number and PIN.

dom(BE) = {AID/rc, AP/rc, TS/g}; the bank employee can read and copy the account ID and password and grant any rights to any subject.

dom(BS) = {AN/rc, AID/rc, AP/rwc, DCN/rc, DCP/rc, AB/rwc, $/rwc, TS/g}; the bank server can read the account number, account ID, password, debit card number, debit card PIN, account balance, and money. It can also write the password, balance, and money. All the preceding rights have a copy flag and the bank server can grant any rights it possesses to any subject.

5.2.3 Links between Entities. Links are established if a ticket for an entity is in the domain of the other entity in the relationship. The mathematical definition of this relation is

$$link(\mathrm{A},\mathrm{B}) \equiv \mathrm{A/k} \in dom(\mathrm{B}) \text{ or } \mathrm{B/k} \in dom(\mathrm{A}) \text{ where A and B are entities and } \mathrm{k} \in \{\text{rights}\}$$

5.2.4 Filters. For the initial analysis, it is assumed the filter function, f, allows for any ticket to be transferred since this is the worst case scenario or $T \times R \in f(u,v)$ where T is the set of all Types and R is the set of all Rights. The initial state is represented graphically as shown in Figure 5.1. Objects are represented with blue circles, subjects with green circles, and links with solid lines. Filters are not shown since all tickets can be transferred based on the initial filter function.

5.2.5 History. Without creating new subjects, the safety of the system can be initially analyzed by creating a history of the transfer of all possible tickets. The filter functions are not specified in each step of the history because they allow all tickets to be transferred.

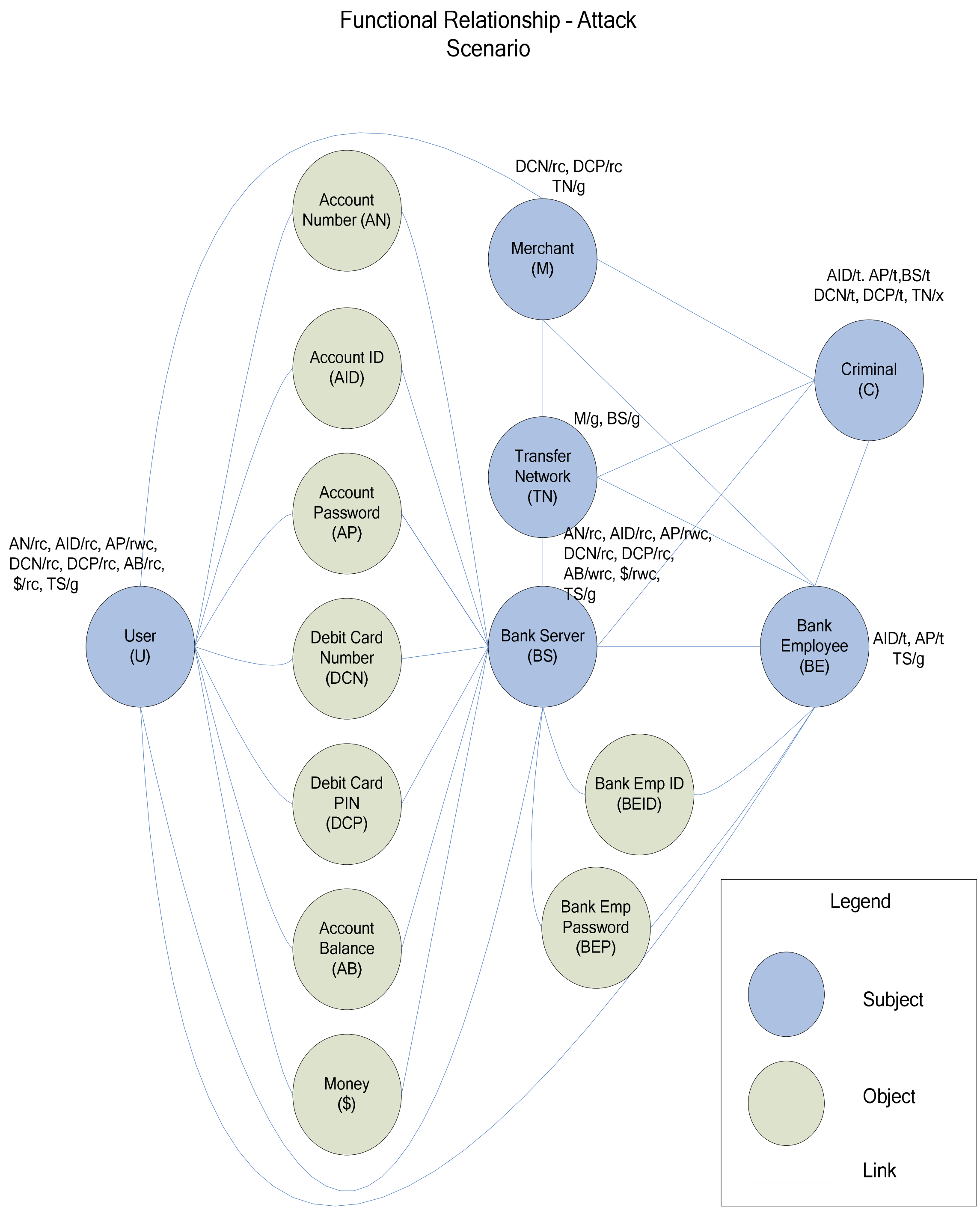

Figure 5.1: Relationships between Objects and Subjects in System.

User transfers all possible tickets to Merchant

link(U,M) = TS/g∈ *dom*(U) is true

dom(M) = {TN/g, AN/rc, AID/rc, AP/rwc, DCN/rc, DCP/rc, AB/rc, $/rc}

User transfers all possible tickets to Bank Server

link(U,BS) = BS/g∈ *dom*(U) is true

dom(BS) = {AN/rc, AID/rc, AP/rwc, DCN/rc, DCP/rc, AB/rwc, $/rwc, TS/g}

In this case, the User does not possess any tickets that the Bank Server does not already possess so its domain remains unchanged.

Bank Server transfers all possible tickets to Transfer Network, Bank Employee, and Criminal

link(BS,TN) = TS/g∈ *dom*(BS) is true

link(BS,BE) = TS/g∈ *dom*(BS) is true

link(BS,C) = TS/g∈ *dom*(BS) is true

dom(BE) = {AN/rc, AID/trc, AP/trwc, DCN/rc, DCP/rc, AB/rwc, $/rwc,TS/g}

dom(TN) = {AN/rc, AID/rc, AP/rwc, DCN/rc, DCP/rc, AB/rwc, $/rwc, TS/g}

dom(C) = {AN/rc, AID/trc, AP/trwc, DCN/trc, DCP/trc, AB/rwc, $/rwc, BS/t, TN/x}

Because links exist between all subjects, and the filters specify all tickets can be transferred over the links, every subject can obtain all tickets in the user's initial domain. The flow function for any two subjects is then the complete list of rights that have a copy

flag in any of the subject's domains. Clearly, the filters must be modified to prevent subjects from obtaining unauthorized tickets.

To determine how to properly implement filters to prevent unauthorized rights from leaking, an attack tree and a protection tree are developed and shown in Figures 5.2 and 5.3. Based on an analysis of the protection tree, to completely protect the network, three areas must be protected. They include preventing unauthorized online access, protecting the bank server, and protecting the transfer network. To protect the bank server, the network is hardened to prevent outside attack and internal transactions are logged. To protect the transfer network a challenge and response system is set up. To protect against unauthorized account access there are many different choices. After conducting an analysis of the protection tree, it is determined that the most efficient means is to implement a challenge and response system as part of a two-factor authentication scheme. Filters are now implemented in the SPM to achieve these protections.

5.3 Extension of Authentication in SPM

Authentication in SPM is addressed by [VaC94] where conditional tickets are introduced. These tickets cannot be exercised unless a specified condition is met. Although for some models this is necessary, it is not sufficient for the current implementation. For instance, some rights might not require authentication to exercise once a subject possesses them and therefore should not be transferred to other subjects without first authenticating. In the banking example, a read right on money means the entity possesses the money. Once the money is transferred, no authentication is

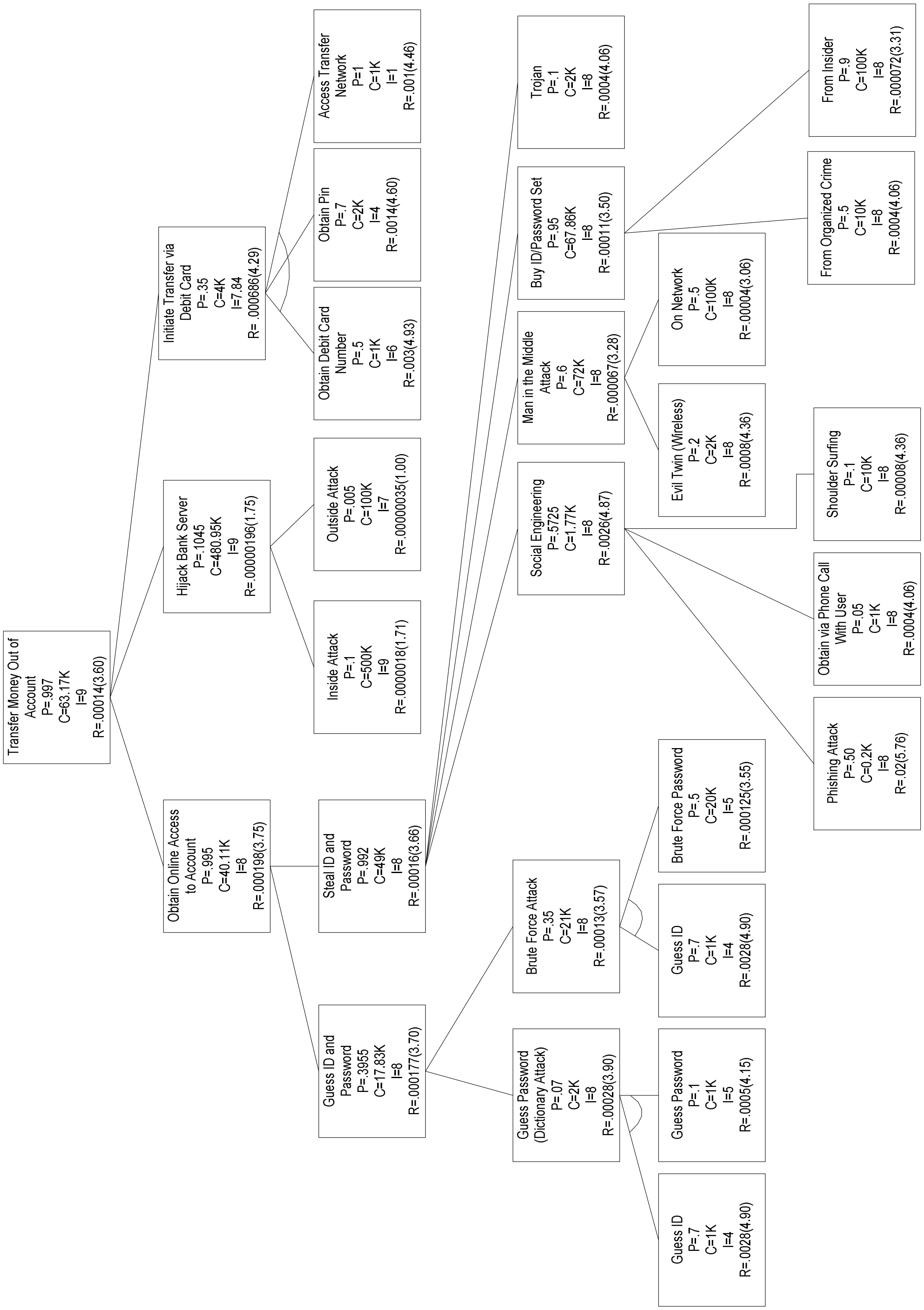

Figure 5.2: Online Banking Attack Tree.

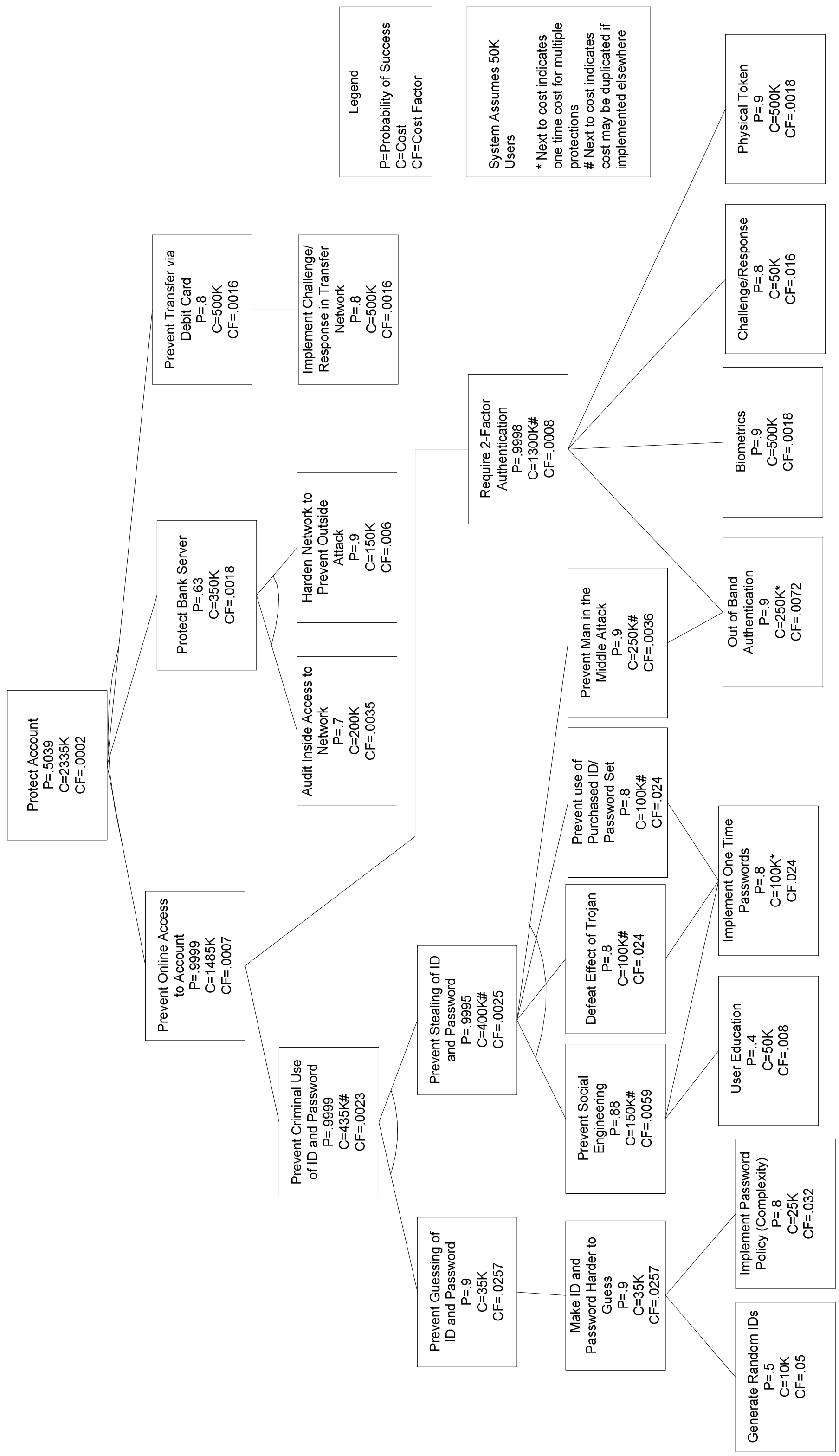

Figure 5.3: Online Banking Protection Tree.

necessary to use it. In this case, authentication should not be attached to the right but rather to the transfer of that right by invoking authentication through predicate logic in the filter function to prevent the copying of tickets to unauthorized subjects.

5.4 Implementing Filters

The following sections use filter functions with predicate logic to implement the protections identified through the analysis of the developed protection tree.

5.4.1 Implement Challenge and Response System with User Accounts.

Communication between the bank and the user can occur *a priori* and out of band in a secure way to set up the challenge and response questions. The answers to the questions are processed with one-way hashes and a "salt" value. The results are stored in the bank server. The salt is an additional input to the key derivation function (KDF) which hashes the responses. This ensures that even if the server is compromised, the original answers to the challenges cannot be determined as they are encrypted with a one-way hash function. It also prevents an adversary from intercepting the hashed value and using a dictionary attack to recover the stored response. When the user authenticates with the server, the predetermined "salt" is added, the response is hashed, and the result is sent to the server which compares it to the stored hash using function g.

$$f(BS,U) = \{AN/rc, AID/rc, AP/rc, DCN/rc, DCP/rc, AB/wrc, \$/rwc\}$$
$$\text{if } g(UR + salt) = g(SR + salt)$$
$$\text{where } UR = \text{ User Response and } SR = \text{ Stored Response}$$

5.4.2 Harden Bank Server to Prevent Outside Attack. Although an actual implementation to harden the bank server is not specified, the desired effect is that no

rights can flow from the bank server to an unauthorized entity. Therefore, the filter is specified that no rights can transfer directly from the bank server to the criminal.

$$f(BS,C)=\varnothing$$

The filter is modeled in this way to represent how the transfer of rights occurs in normal operations. If the bank server is hardened, a criminal cannot force the server to operate in an unintended manner thus giving up rights. Essentially, the criminal cannot "hack" the hardened server to gain information. He must instead use some type of legitimate transfer of rights to try and gain access.

5.4.3 Audit Inside Access to Network. Auditing does not prevent any inside attack but allows retribution after the fact. At best, it serves as a deterrent. As such, it is not implemented in this SPM model.

5.4.4 Implement Challenge and Response System in Transfer Network. This is similar to the challenge and response system set up between the user and the bank server. All parties in the transfer network must set up appropriate challenges and responses *a priori* to the transaction. This is assumed to be done over secure communications or a separate out of band communication system so the information is not compromised. Only hashed values are stored to protect the information from a later compromise. The filter ensures that rights are transferred contingent on a successful challenge and response.

$$f(TN,TS)=\{DCN/rc, DCP/rc, \$/rwc\}$$
if $g(UR+salt)=g(SR+salt)$
where $UR=$ User Response and $SR=$ Stored Response

5.4.5 History with Filters Enabled. Now that the filters implement the desired

protections, a new history is developed to test the leakage of rights in the system.

5.4.5.1 User transfers all possible tickets to Merchant.

link(U,M) = TS/g$\in$ *dom*(U) is true

$f(U,M) = T \times R$

dom(M) = {AN/rc, AID/rc, AP/rwc, DCN/rc, DCP/rc, AB/rc, $/rc, TN/g}

5.4.5.2 Merchant transfers all possible tickets to Criminal. In this case, the Criminal mimics the Transfer Network to get the User's tickets so the links and filters that define the relationship between the Transfer Network and the Merchant are used.

link(M,C) = *link*(M,TN)=TN/x$\in$ *dom*(C) is true

$f(TN,TS) = \{DCN/rc, DCP/rc, \$/rwc\}$
if $g(UR + salt) = g(SR + salt)$
where $UR =$ User Response and $SR =$ Stored Response

Because the Criminal cannot successfully complete the challenge and response with the Merchant, no additional tickets are transferred.

dom(C) = {AID/t, AP/t, BS/t, DCN/t, DCP/t, TN/x}

5.4.5.3. User transfers all possible tickets to Bank Server.

link(U,BS)=TS/g$\in$ *dom*(U) is true

$f(BS,U) = AN/rc, AID/rc, AP/rc, DCN/rc, DCP/rc, AB/wrc, \$/rwc$
if $g(UR + salt) = g(SR + salt)$
where $UR =$ User Response and $SR =$ Stored Response

dom(BS) = {AN/rc, AID/rc, AP/rwc, DCN/rc, DCP/rc, AB/rwc, $/rwc, TS/g}

As in the original history, the Bank Server does not gain any additional tickets because it already had all of the User's tickets in its domain.

5.4.5.4 Bank Server transfers all possible tickets to Transfer Network.

link(BS,TN) = TS/g$\in$ *dom*(BS) is true

$$f(TN, TS) = \{DCN/rc, DCP/rc, \$/rwc\}$$
if $g(UR + salt) = g(SR + salt)$
where $UR =$ User Response and $SR =$ Stored Response

Because the Transfer Network and the Bank Server can complete a successful challenge and response, all tickets specified in the filter may be transferred and the Transfer Network domain becomes:

dom(TN) = {DCN/rc, DCP/rc, $/rwc}

5.4.5.5 Bank Server transfers all possible tickets to Criminal.

link(BS,C) = TS/g$\in$ *dom*(BS) is true

$$f(BS, C) = \varnothing$$

Because the filter consists of the null set, no rights may be transferred between the Bank Server and the Criminal. Therefore, the Criminal cannot gain any new rights. If the Criminal acts as a User, Merchant, or Transfer Network, the challenge and response cannot be successful. Therefore, no additional rights can be transferred as specified by the appropriate filters.

dom(C) = {AID/t, AP/t, BS/t, DCN/t, DCP/t, TN/x}

Based on the tickets the Criminal possesses, he cannot get the money from the Bank Server.

5.4.5.6 Criminal takes all possible tickets from the Transfer Network.

link(M,C) = TN/x$\in$ *dom*(C) is true

$$f(TN, TS) = \{DCN / rc, DCP / rc, \$ / rwc\}$$
if $g(UR + salt) = g(SR + salt)$
where $UR =$ User Response and $SR =$ Stored Response

The Criminal cannot complete a successful challenge and response with the Transfer Network so the Criminal does not gain any additional tickets.

dom(C) = {AID/t, AP/t, BS/t, DCN/t, DCP/t, TN/x}

5.4.5.7 Bank Server grants all possible rights to Bank Employee.

link(BS,BE) = BE/g∈ *dom*(BS) is true

$$f(BS, BE) = \{AN / rc, AID / rc, AP / rc, DCN / rc, DCP / rc, AB / wrc, \$ / rwc\}$$
if $g(BER + salt) = g(SR + salt)$
where $BER =$ Bank Employee Response and $SR =$ Stored Response

dom(BE) = {AN/rc, AID/trc, AP/trwc, DCN/rc, DCP/rc, AB/rwc, $/rwc, TS/g}

5.4.5.8 Bank Employee grants all possible rights to Criminal.

A serious threat to any system is the insider threat. Should an insider provide critical information to a Criminal either intentionally or accidentally through social engineering, many of the protections can be easily bypassed. In this example, it is assumed the Bank Employee has all customer information but does not have access to the challenge and response system used by the Bank Server.

link(BE,C)=TS/g∈ *dom*(BE) is true

$$f(BE, TS) = \{AN / rc, AID / rc, AP / rc, DCN / rc, DCP / rc, AB / wrc\}$$

The Bank Employee cannot transfer the money to the criminal because the criminal must use either the Transfer Network or the Bank Server to do this. The Employee could transfer the money into another account which cannot be prevented by these protections but would be discovered through auditing.

dom(BE) = {AN/rc, AID/trc, AP/trwc, DCN/rc, DCP/rc, AB/rwc, $/rwc, TS/g}

dom(C) = {AN/rc, AID/rct, AP/rct, AB/rwc, BS/t, DCN/rct, DCP/rct, TN/x}

The final set of domains is:

dom(U) = {AN/rc, AID/rc, AP/rwc, DCN/rc, DCP/rc, AB/rc, $/rc, TS/g}

dom(BS) = {AN/rc, AID/rc, AP/rwc, DCN/rc, DCP/rc, AB/rwc, $/rwc, TS/g}

dom (M) ={AN/rc, AID/rc, AP/rwc, DCN/rc, DCP/rc, AB/rc, $/rc, TN/g}

dom (TN) = {DCN/rc, DCP/rc, $/rwc}

dom(BE) = {AN/rc, AID/trc, AP/trwc, DCN/rc, DCP/rc, AB/rwc, $/rwc, M/g, TS/g}

dom(C) = {AN/rc, AID/rct, AP/rct, AB/rwc, BS/t, DCN/rct, DCP/rct, TN/x}

5.5 Analysis

Based on the final set of domains, it is apparent only subjects who have legitimate needs to have rights over the money in the account do. The criminal cannot obtain these rights due to the filters that implement the protections identified in the system's protection tree. This maximal state shows that the protection tree results in a safe system with respect to the rights (i.e. read rights on User's money) that are being protected.

Although the Criminal does possess critical information, he cannot get the money via the Merchant, Transfer Network, or Bank Server because he cannot successfully complete a challenge and response with the associated subject. As that is the goal of the stated protections with these subjects, the protections are shown to be successful.

This analysis intentionally does not utilize the create operation of SPM. The reason for this is even if new subjects are created, due to the attenuating create rules, no

subjects would have more rights than the subjects that created them. The new subjects would also be subject to the same filters between subject types and thus would not result in any different sets of domains.

5.6 Summary

After modeling the online banking system with the SPM, the transfer of rights in a system with protections implemented is compared against the baseline system without protections. It is determined that the system without protections is unsafe with respect to the read right over the money which implies the Criminal can get unauthorized access to the User's money. After analyzing the system's vulnerabilities with an attack tree, three intermediate vulnerabilities are determined to exist. The protection tree is then developed and analyzed finding that three different types of protections are necessary to adequately protect the system. These protections are implemented using the conditional filters from the extended SPM model. After a complete analysis, the maximal flow is determined for the system and the system is deemed a safe system with respect to the read right on the User's money. By showing that the identified protections result in a safe system using the SPM, the effectiveness of the attack and protection tree methodology is shown.

VI. Conclusion

Today's critical systems are under constant threat of attack from a variety of adversaries. To properly protect these systems with limited resources, the security practitioners need a methodology to ensure that the protections implemented to mitigate the vulnerabilities in the systems are the most efficient ones possible. The methodology developed out of this research effort does that.

6.1 Research Contributions

The primary contribution of this research is the development of an attack and protection tree methodology that can analyze the security of complex systems. This methodology is an extension of the existing concept of attack trees but significantly improves that concept as well as developing the new concept of protection trees.

This research also makes significant strides by using security metrics within the attack and protection tree security analysis. Rule sets are developed and tested in a variety of applications. Both single objective optimization problems as well as multi-objective optimization problems can be analyzed using the constructs of the framework.

The developed framework can be generalized to be useful in a multitude of application domains. Specifically, the methodology is employed to analyze general computer networks, a DHS information sharing network, online banking, and MANETs. Thus, this framework can be adapted for use in any complex system that has defenders, attackers, and resources to protect. It is this generalization of the methodology that makes it useful to many different disciplines.

6.1.1 Related Research Contributions. In developing the framework,

validation of the methodology is important. This is accomplished using an extended version of the Schematic Protection Model (SPM). Specifically, the method of authentication in SPM is changed to allow authentication on the transfer of a right rather than on the use of that right. This extension properly models the protection for the subject systems. The new authentication model has applications in other areas of research beyond the methodology presented here such as physical security and natural disaster planning.

In addition to the actual methodology presented, additional research into vulnerability discovery and mitigation includes using the protections found in viruses for legitimate applications. A general taxonomy of the protections that viruses employ is developed that allows researchers in the software protection field determine how they can protect critical applications using novel means.

To show how some of the virus protections could actually be used, metamorphic code is implemented in experiments. These experiments show that metamorphism used in legitimate applications have good results. Specific measurements are also presented on how effective different metamorphic schemes interfere with the debugging of the programs.

Computer viruses are also explored with respect to detection methods. From this research, a new evolutionary algorithm (REALGO) is created. This effort successfully demonstrates that the REALGO algorithm indeed performs better than others in its class. In addition, the algorithm is used by another researcher in an IDS implementation with very good results [Haa07].

Additional research is conducted on the intrusion detection systems of MANETs. This IDS has a multi-objective component due to its requirement to be efficient but also must have a high detection rate. This research chronicles the current research in this area as well as proposing how a multi- objective artificial immune system (MOAIS) algorithm is implemented in an IDS for a MANET.

6.1.2 Publications. The number of refereed publications in international journals and conference proceedings demonstrates the novelty of this research. To date, eight papers have been published including two journal articles. An additional four papers are under review including three additional journal articles. Specific titles and publication venues are listed at the end of this chapter.

6.2 Recommendations for Future Research

Future work includes the application of the attack and protection tree methodology in a variety of different domains to include physical security situations and natural disaster planning and recovery. More specifically, homeland security applications are a prime area for additional research. Future research should also include automating the process of analyzing a system using attack and protection trees. An example of this type of automation is shown in Figure 6.1. This figure represents how a software protection system would be automated using an integrated compiler with standard libraries of attacks and protections. Libraries of standard attacks and protections would obviously need to be created to keep the problems scalable. Methods of keeping these libraries updated also need to be developed.

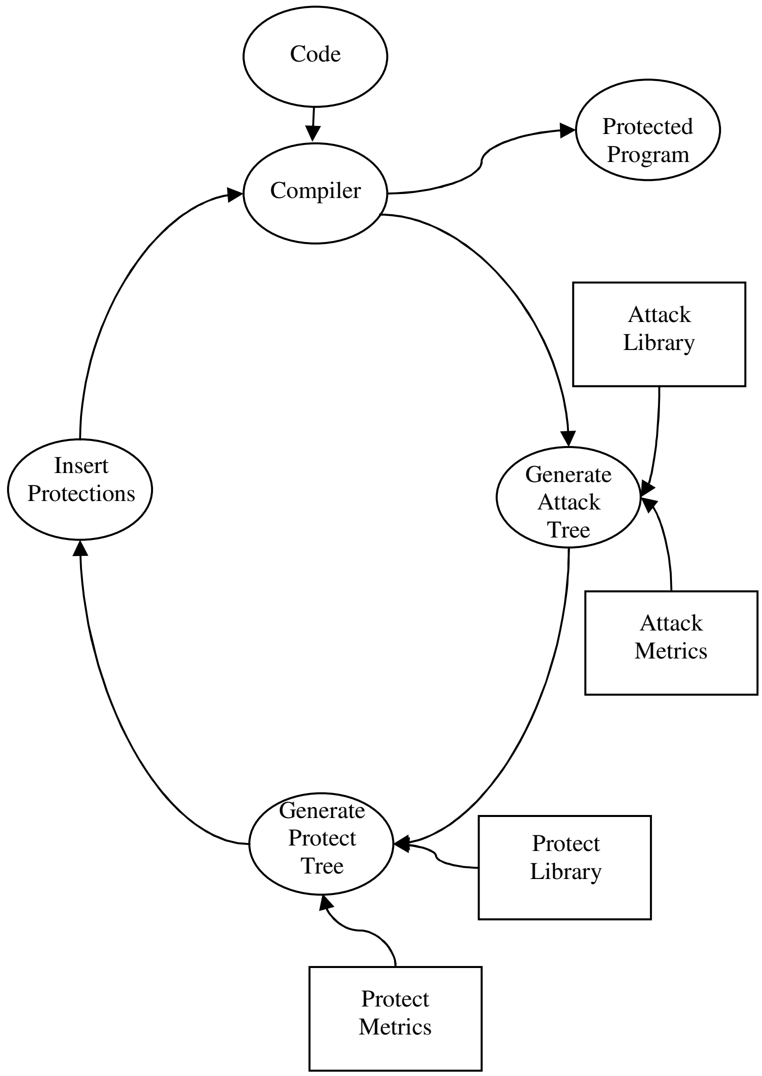

Figure 6.1: Automated Protection System for Software Applications.

In general, this research explores a new area of attack modeling and mitigation techniques. As such, the opportunity for additional research in a multitude of different areas of the attack and protection tree methodology is great. The framework presented here is the foundation for the security solutions of the future.

Publications

Published (8):

[EdD06a] Edge, K., T. Dube, R. Raines, R. Baldwin, B. Mullins, and C. Reuter, "A Taxonomy of Protections Used in Computer Viruses and Their Applications to Software Protection," *Proceedings of the International Conference on i-Warfare and Security*, 67-76. University of Maryland Eastern Shore, MD, 2006.

[DuE06] Dube, T., K. Edge, R. Raines, R. Baldwin, B. Mullins, and C. Reuter, "Metamorphism: A Software Protection Mechanism," *Proceedings of the International Conference on i-Warfare and Security*, 57-66. University of Maryland Eastern Shore, MD, 2006.

[EdL06a] Edge, K. S., G. B. Lamont, and R. A. Raines, "Multi-Objective Mobile Network Anomaly Intrusion" *International Journal of Computer Science and Network Security*, vol. 6 no. 3B, pp. 187-192, 2006.

[EdL06b] Edge, K. S., G. B. Lamont, and R. A. Raines, "A Retrovirus Inspired Algorithm for Virus Detection & Optimization," *Proceedings of the IEEE Genetic and Evolutionary Computation Conference*, 103-110. Seattle, WA, 2006.

[EdD06b] Edge, K., G. Dalton, R. Raines, and R. Mills, "Using Attack and Protection Trees to Analyze Threats and Defenses to Homeland Security," *Proceedings of the Military Communications Conference (MILCOM)*, Washington, D.C., 2006.

[EdR07a] Edge, K., R. Raines, M. Grimaila, R. Baldwin, R. Bennington, and C. Reuter, "The Use of Attack and Protection Trees to Analyze Security for an Online Banking System," *Proceedings of the Fortieth Annual Hawaii International Conference on System Sciences (CD-ROM)*, 8 Pages. Waikoloa, Hawaii, 2007.

[EdR07b]Edge, K., R. Raines, C. Reuter, R. Bennington, R. Baldwin, and M. Grimaila, "Analyzing Security Measures for Mobile Ad Hoc Networks Using Attack and Protection Trees," *Proceedings of the International Conference on Information Warfare and Security*, Monterey, CA, 2007.

[EdR07c] Edge, K., R. Raines, C. Reuter, R. Bennington, R. Baldwin, and M. Grimaila, "Analyzing Security Measures for Mobile Ad Hoc Networks Using Attack and Protection Trees," To Appear *in the Journal of Information Warfare*, 2007.

Under Review (4):

Kenneth Edge, Richard Raines, Rusty Baldwin, Michael Grimaila, Christopher Reuter, and Robert Bennington. "Extending the Schematic Protection Model to Verify the Safety of a System Using Attack and Protection Trees," Under review by *IEEE Transactions on Information Forensics and Security*.

Kenneth Edge, George Dalton, Richard Raines, and Robert Mills. "Analyzing Network Security Using Attack and Protection Trees" Under review by *Computers and Security.*

Kenneth Edge, Richard Raines, Christopher Reuter, Robert Bennington, Rusty Baldwin, and Michael Grimaila. "The Use of Attack and Protection Trees to Analyze Security for an Online Banking System," Under review by *IEEE Transactions on Systems, Man, and Cybernetics – Part A.*

Kenneth Edge, Richard Raines, Christopher Reuter, Robert Bennington, Rusty Baldwin, and Michael Grimaila. "Analyzing Tradeoffs Between Cost And Performance For Mobile Ad Hoc Network Security Solutions Using Attack And Protection Trees," Submitted to *The Ninth IFIP/IEEE International Conference on Mobile and Wireless Communications Networks.*

Bibliography

[AlC02] Albers, P., O. Camp, J. Percher, B. Jouga, L. Me, and R. Puttini, "Security in Ad Hoc Networks: A General Intrusion Detection Architecture Enhancing Trust Based Approaches," *Proceedings of the 1st International Workshop WL Information Systems, 4th International Conference on Enterprise Information Systems,* Ciudad Real, Spain, 2002.

[Ame00] "Metamorphosis," in *American Heritage Dictionary of the English Language* Fourth ed: Houghton Mifflin Company 2000.

[Ame05] Amenaza, T. L., "Fundamentals of Capabilities-based Attack Tree Analysis," Calgary, Canada, November 8, 2005.

[Ame06a] Amenaza, T. L., "Advanced Attack Tree Based Risk Analysis," Calgary, Canada, February 16, 2006.

[Ame06b] Amenaza, T. L. SecurITree. Ver. 2.4. Computer Software. www.amenaza.com, 2006.

[Amo94] Amoroso, E. G., *Fundamentals of Computer Security Technology*. Englewood Cliffs, NJ: P T R Prentice Hall, 1994.

[AmS90] Ammann, P. E. and R. S. Sandhu, "Extending the Creation Operation in the Schematic Protection Model," *Proceedings of the Sixth Annual Computer Security Application Conference,* pp. 340-348. Tucson, AZ, 1990.

[AmS91] Ammann, P. E. and R. S. Sandhu, "Safety Analysis for the Extended Schematic Protection Model," *Proceedings of the IEEE Symposium on Research in Security and Privacy,* pp. 87-97. Oakland, CA, 1991.

[Bac96] Bäck, T., *Evolutionary Algorithms in Theory and Practice*. New York: Oxford University Press, 1996.

[BiF06] Bistarelli, S., F. Fioravanti, and P. Peretti, "Defense Trees for Economic Evaluation of Security Investments," *Proceedings of the First International*

Conference on Availability, Reliability and Security (ARES'06), pp. 416-423. Vienna, Austria, 2006.

[Bis03] Bishop, M., *Computer Security Art and Science*. Boston, MA: Addison-Wesley, 2003.

[Bla96] Blade_Runner_(alias). "Encryption: Practice." *29A Ezine*. http://www.madchat .org/vxdevl/vxmags/29a-1/29A%231.2_2. Accessed 14 Aug 2005.

[BrP03] Brooke, P. J. and R. F. Paige, "Fault Trees For Security System Design and Analysis," *Computers & Security*, vol. 22, no. 3, pp. 256-264, 2003.

[CaT02] Castro, L. d. and J. Timmis, *Artificial Immune Systems: A New Computational Intelligence Approach*. London: Springer-Verlag, 2002.

[Cer05] www.cert.org. Accessed 16 Aug 2005.

[CoT97] Collberg, C., C. Thomborson, and D. Low, "A Taxonomy of Obfuscating Transformations," Department of Computer Science, University of Auckland, 1997.

[DaH04] Dawkins, J. and J. Hale, "A Systematic Approach to Multi-stage Network Attack Analysis," *Proceedings of the Second IEEE International Information Assurance Workshop,* pp. 48 - 56. Charlotte, NC, 2004.

[DaL02] Daley, K., R. Larson, and J. Dawkins, "A Structural Framework for Modeling Multi-stage Network Attacks," *Proceedings of the 2002 International Conference on Parallel Processing Workshops (ICPPW'02),* pp. 5-10. Vancouver, B.C., Canada, 2002.

[DaM05] da Silva, A., M. Martins, B. Rocha, A. Loureiro, L. Ruiz, and H. Wong, "Decentralized Intrusion Detection in Wireless Sensor Networks," *Proceedings of the 1st ACM International Workshop on Quality of Service & Security in Wireless and Mobile Networks,* Montreal, Canada, 2005.

[Das02] Dasgupta, D., "An Immunity-Based Technique to Characterize Intrusions in Computer Networks," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 3, 2002.

[Das99] Dasgupta, D., *Artificial Immune Systems and Their Applications*. New York: Springer, 1999.

[Dat05] DataRescue. "The IDA Pro Disassembler and Debugger." http://www.datarescue.com/idabase. Accessed 1 Dec 2005.

[DeL02] Deng, H., W. Li, and D. Agrawal, "Routing Security in Wireless Ad Hoc Networks," in *IEEE Communications Magazine*, October 2002, pp. 70-75.

[DoH01] Downs, D. D. and R. Haddad, "Penetration Testing – The Gold Standard For Security Rating and Ranking," *Proceedings of the 1st Workshop on Information-Security-System Rating and Ranking (WISSRR),* Williamsburg, VA, 2001.

[DuE06] Dube, T., K. Edge, R. Raines, R. Baldwin, B. Mullins, and C. Reuter, "Metamorphism: A Software Protection Mechanism," *Proceedings of the International Conference on i-Warfare and Security,* pp. 57-66. University of Maryland, Eastern Shore, MD, 2006.

[EdD06a] Edge, K., T. Dube, R. Raines, R. Baldwin, B. Mullins, and C. Reuter, "A Taxonomy of Protections Used in Computer Viruses and Their Applications to Software Protection," *Proceedings of the International Conference on i-Warfare and Security,* pp. 67-76. University of Maryland, Eastern Shore, MD, 2006.

[EdD06b] Edge, K., G. Dalton, R. Raines, and R. Mills, "Using Attack and Protection Trees to Analyze Threats and Defenses to Homeland Security," *Proceedings of the Military Communications Conference (MILCOM),* Washington, D.C., 2006.

[EdL06a] Edge, K. S., G. B. Lamont, and R. A. Raines, "Multi-objective Mobile Network Anomaly Intrusion," *International Journal of Computer Science and Network Security*, vol. 6, no. 3B, pp. 187-192, 2006.

[EdL06b] Edge, K. S., G. B. Lamont, and R. A. Raines, "A Retrovirus Inspired Algorithm for Virus Detection & Optimization," *Proceedings of the IEEE Genetic and Evolutionary Computation Conference,* pp. 103-110. Seattle, WA, 2006.

[EdR07a] Edge, K., R. Raines, M. Grimaila, R. Baldwin, R. Bennington, and C. Reuter, "The Use of Attack and Protection Trees to Analyze Security for an Online Banking System," *Proceedings of the Fortieth Annual Hawaii International Conference on System Sciences (CD-ROM),* Waikoloa, Hawaii, 2007.

[EdR07b] Edge, K., R. Raines, C. Reuter, R. Bennington, R. Baldwin, and M. Grimaila, "Analyzing Security Measures for Mobile Ad Hoc Networks Using Attack and Protection Trees," *Proceedings of the International Conference on Information Warfare and Security,* Monterey, CA, 2007.

[EdR07c] Edge, K., R. Raines, C. Reuter, R. Bennington, R. Baldwin, and M. Grimaila, "Analyzing Security Measures for Mobile Ad Hoc Networks Using Attack and Protection Trees," *To Appear in the Journal of Information Warfare*, 2007.

[Eil05] Eilam, E., *Reversing: Secrets of Reverse Engineering*. Indianapolis: Wiley Publishing, 2005.

[Erd03] Erdelyi, G. "Hide 'n' Seek? Anatomy of Stealth Malware." *Virus Bulletin*. http://www.google.com/url?sa=t&ct=res&cd=1&url=http%3A//www.blackhat.com/presentations/bh-europe-04/bh-eu-04-erdelyi/bh-eu-04-erdelyi-paper.pdf &ei=kowdQ7OaN8L8iQHW08mKCw Accessed 4 Sep 2005.

[Esh00] Eshelman, L., "Genetic Algorithms," in *Evolutionary Computation 1: Basic Algorithms and Operators*, Bäck, Fogel, and Michalewicz, Eds. Bristol: Institute of Physics, pp. 64-80, 2000.

[FFI06] "Authentication in an Internet Banking Environment." http://www.ffiec.gov/pdf/authentication_guidance.pdf. Accessed 24 May 2006.

[FoH00] Forrest, S. and S. Hofmeyr, "Immunology as Information Processing," in *Design Principles for Immune System & Other Distributed Autonomous Systems*, Segel and Cohen, Eds. Oxford: University Press, pp. 361-387, 2000.

[Fox05] Fox, S. "Pew Internet & American Life Project Survey." http://www.pewinternet.org/pdfs/PIP_Online_Banking_2005.pdf. Accessed 22 May 2006.

[Fse05] "F-Secure Virus Descriptions." http://www.f-secure.com/v-descs. Accessed 15 Nov 2005.

[Gar05] Garrett, S. M., "How Do We Evaluate Artificial Immune Systems?," *Evolutionary Computation*, vol. 13, no. 2, pp. 145-178, 2005.

[GiC05] Giffin, J., M. Christodorescu, and L. Kruger, "Strengthening Software Self-Checksumming via Self-Modifying Code," Computer Sciences Department, University of Wisconsin, 2005.

[GoF96] Gordon, S. and R. Ford, "Real World Anti-virus Product Reviews and Evaluations – The Current State of Affairs," *Proceedings of the 9th National Information Systems Security Conference (NISSC'96),* pp. 526-538. Baltimore, MD, 1996.

[Gri04] Grimes, B., "Homeland Security gets in the Groove," in *Washington Technology*, vol. 19, 2004.

[Gri97] GriYo_(alias). "Gollum Virus." *29A Ezine*. http://www.madchat.org/vxdevl /vxmags/29a-2/29A%232.3_A. Accessed 21 Aug 2005.

[Gro06] "Groove Virtual Office at a Glance." http://www.groove.net/pdf/gaag/gaag-architecture.pdf. Accessed 15 May 2007.

[Haa07] Haag, C., "An Artificial Immune System-Inspired Multiobjective Evolutionary Algorithm With Application To The Detection Of Distributed Computer Network Intrusions", Master of Science Thesis: Air Force Institute of Technology, 2007.

[HaG03] Hariri, S., Q. Guangzhi, T. Dharmagadda, M. Ramkishore, and C. S. Raghavendra, "Impact Analysis of Faults and Attacks in Large-scale Networks," *Security & Privacy Magazine, IEEE*, vol. 1, no. 5, pp. 49-54, 2003.

[HaL00] Hammer, P. K. and G. B. Lamont, "An Agent based Architecture for a Computer Virus Immune Systems," *Proceedings of the Artificial Immune System Workshop of the Genetic and Evolutionary Computation Conference,* Las Vegas, NV, 2000.

[HaW02] Harmer, P. K., P. D. Williams, C. H. Gunsch, and G. B. Lamont, "An Artificial Immune System Architecture for Computer Security Applications," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 3, pp. 252-280, 2002.

[HoF00] Hofmeyer, S. and S. Forrest, "Architecture for an Artificial Immune System," *Evolutionary Computation*, vol. 8, no. 4, pp. 443-473, 2000.

[Hof99] Hofmeyer, S., "An Immunological Model of Distributed Detection and Its Application to Computer Security", PhD Dissertation: University of New Mexico, 1999.

[Int05] Intel, *IA-32 Intel Architecture Software Developer's Manual*, vol. 2B: Intel Corporation, 2005.

[Jac97a] Jack_Qwerty_(alias). "Win32.Cabanas.2999." *29A Ezine*. http://www.madchat.org/vxdevl/vxmags/29a-2/29A%232.3_6. Accessed 14 Aug 2005.

[Jac97b] Jack_Qwerty_(alias). "Win32.Jacky.1440." *29A Ezine*. http://www.madchat.org/vxdevl/vxmags/29a-2/29A%232.3_7. Accessed 21 Aug 2005.

[KaG03] Kachirski, O. and R. Guha, "Effective Intrusion Detection Using Multiple Sensors in Wireless Ad Hoc Networks," *Proceedings of the 36th Hawaii International Conference on System Sciences,* Waikoloa, Hawaii, 2003.

[Kar05] Karppinen, K. "Security Measurement Based on Attack Trees in a Mobile Ad Hoc Network Environment." http://www.vtt.fi/inf/pdf/publications/2005/P580.pdf. Accessed 15 May 2007.

[KaS05] Kasslin, K., M. Stahlberg, S. Larvala, and A. Tikkanen, "Hide 'N Seek Revisited – Full Stealth is Back," *Proceedings of the Virus Bulletin Conference,* Dublin, Ireland, 2005.

[Key05] Keynote Systems, I. "Online Banking Critical to Bank Selection and Brand Perception." 6 Jan 2005. http://www.keynote.com/news_events/releases_2005/05jan06.html. Accessed 23 May 2006.

[Kle03] Kleiboeker, S. B., "Applications of Competitor RNA in Diagnostic Reverse Transcription-PCR," *Journal of Clinical Microbiology*, vol. 41, no. 5, pp. 2055-2061, 2003.

[Koh95] Kohavi, R., "A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection," *Proceedings of the International Joint Conferences on Artificial Intelligence,* pp. 1137-1143. 1995.

[LaB94] Landwehr, C. E., A. R. Bull, J. P. McDermott, and W. S. Choi, "A Taxonomy of Computer-Program Security Flaws," *ACM Computing Surveys*, vol. 26, no. 3, pp. 211-254, 1994.

[LaM99] Lamont, G. B., R. Marmelstein, and D. V. Veldhuizen, "A Distributed Architecture for a Self-Adaptive Computer Virus Immune System," in *New Ideas in Optimization*, D. Corne, and Glover, Ed. Maidenhead, England: McGraw Hill 1999.

[LaS04] Langweg, H., E. Snekkenes, H. Hassanein, R. L. Oliver, G. G. Richard, L. F. Wilson, H. Hassanein, R. L. Oliver, G. G. Richard, and L. F. Wilson, "A Classification of Malicious Software Attacks," *Proceedings of the 2004 IEEE International Performance, Computing, and Communications Conference,* pp. 827-832. Piscataway, NJ, 2004.

[LiM01] Linger, R. C. and A. P. Moore, "Foundations for Survivable System Development: Service Traces, Intrusion Traces, and Evaluation Models," Carnegie Melon, Software Engineering Institute CMU/SEI-2001-TR-029, 2001.

[LiR98] Liggesmeyer, P. and M. Rothfelder, "Improving System Reliability With Automatic Fault Tree Generation," *Proceedings of the Twenty-Eighth Annual International Symposium on Fault-Tolerant Computing,* pp. 90-99. Munich, Germany, 1998.

[Lor97] Lord_Julus_(alias). "Analysis on the Decryptor Generation." *29A Ezine*. http://www.madchat.org/vxdevl/vxmags/29a-2/29A%232.2_9. Accessed 21 Aug 2005.

[Low76] Lowrance, W. W., *Of Acceptable Risk*. Los Altos, CA: William Kauffmann, 1976.

[MaO05] Mauw, S. and M. Oostdijk, "Foundations of Attack Trees," *Proceedings of the Eighth Annual International Conference on Information Security and Cryptology,* pp. 186-198. Seoul, Korea, 2005.

[MaP02] Manikopoulos, C. and S. Papavassiliou, "Network Intrusion and Fault Detection: A Statistical Anomaly Approach," in *IEEE Communications Magazine*, October 2002, pp. 76-82.

[MaW01] Man, M. and V. Wei, "A Taxonomy for Attacks on Mobile Agent," *Proceedings of the International Conference on Trends in Communications,* pp. 385-388. Bratislava, Slovakia, 2001.

[Men04] Mencer, C. S. "ODP Information Bulletin No. 139." October 28, 2004. http://www.ojp.usdoj.gov/odp/docs/info139.htm. Accessed 15 May 2007.

[MoE01] Moore, A. P., R. J. Ellison, and R. C. Linger, "Attack Modeling for Information Security and Survivability," Software Engineering Institute, Carnegie Mellon University, PA, CMU/SEI-2001-TN-001, March, 2001.

[MoK01] Moore, R. A., D. L. Kewley, R. C. Parks, and L. S. Tinnel, "The Information Battlespace Preparation Experiment," *Proceedings of the DARPA Information Survivability Conference & Exposition II, DISCEX '01,* pp. 352-366. Anaheim, California, 2001.

[PaP05] Patwardhan, A., J. Parker, A. Joshi, A. Karygiannis, and M. Iorga, "Secure Routing and Intrusion Detection in Ad Hoc Networks," *Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications,* Kauaii Island, Hawaii, 2005.

[Pay01] Payne, S. C. "A Guide to Security Metrics." July 11, 2001. http://www.sans.org/rr/whitepapers/auditing/55.php. Accessed 8 Feb 2006.

[PeF04] Pettey, C. and S. Friedman. "Gartner Study Shows Thefts from Unauthorized Access to Checking Accounts Resulted in $2.4 Billion in Direct Fraud Losses for U.S. Banks in Past 12 Months." http://www.gartner.com/5_about/press_releases/asset_89228_11.jsp. Accessed 31 May 2006.

[PeP93] J. Percus, O. P., A. Perelson, "Predicting the Size of the T-Cell Receptor and Antibody Combining Region from Consideration of Efficient Self-Nonself Discrimination," *Journal of the National Academy of Sciences of the United States of America*, vol. 90, no. 5, pp. 1691-1695, 1993.

[PoM04] Pozo, R. and B. Miller. "SciMark 2.0." http://math.nist.gov/scimark2. Accessed 1 Dec 2005.

[PrT99] Prowell, S., C. Trammell, R. Linger, and J. Poore, *Cleanroom Software Engineering: Technology and Process*. Reading, MA: Addison-Wesley-Longman, Inc., 1999.

[SaB03] Sarafijanovic, S. and J. Boudec, "An Artificial Immune System Approach with Secondary Response for Misbehavior Detection in Mobile Ad-Hoc Networks," EPFL-DI-ICA, Lausanne, Switzerland TechReport IC/2003/65, 2003.

[SaB05] Sarafijanovic, S. and J. Boudec, "An Artificial Immune System for Misbehavior Detection in Mobile Ad-Hoc Networks with Virtual Thymus, Clustering, Danger Signal, and Memory Detectors," *International Journal of Unconventional Computing*, vol. 1, pp. 221-254, 2005.

[San88] Sandhu, R. S., "The Schematic Protection Model: Its Definition and Analysis for Acyclic Attenuating Schemes," *Journal of the Association for Computing Machinery*, vol. 35, no. 2, pp. 404-432, 1988.

[San89] Sandhu, R. S., "The Demand Operation in the Schematic Protection Model," *Information Processing Letters*, vol. 32, no. 4, pp. 213-219, 1989.

[San92] Sandhu, R., "Undecidability of Safety for the Schematic Protection Model with Cyclic Creates," *Journal of Computer and System Sciences*, vol. 44, pp. 141, 1992.

[SaS98] Salter, C., O. S. Saydjari, B. Schneier, and J. Wallner, "Toward a Secure System Engineering Methodology," *Proceedings of the 1998 Workshop on New Security Paradigms,* pp. 2-10. Charlottesville, Virginia, 1998.

[SaS98] Salter, C., O. S. Saydjari, B. Schneier, and J. Wallner, "Toward a Secure System Engineering Methodolgy," *Proceedings of the 1998 Workshop on New Security Paradigms,* pp. Pages. Charlottesville, Virginia, 1998.

[ScE01] Schultz, M., E. Eskin, E. Zadok, and S. Stolfo, "Data mining Methods for Detection of New Malicious Executables," *Proceedings of the IEEE Symposium on Security and Privacy,* Oakland, CA, 2001.

[Sch00] Schneier, B., *Secrets & Lies: Digital Security in a Networked World*. New York: John Wiley & Sons, 2000.

[Sch01] Schneider, E. A., "Measurements of System Security," *Proceedings of the 1st Workshop on Information-Security-System Rating and Ranking (WISSRR),* Williamsburg, VA, 2001.

[Sch03] Schneier, B., *Beyond Fear: Thinking Sensibly About Security in an Uncertain World*. New York: Copernicus Books, 2003.

[Sch99] Schneier, B. "Modeling Security Threats." *Dr. Dobbs Journal*. December 1999. http://www.schneier.com/paper-attacktrees-ddj-ft.html. Accessed 8 Feb 2006.

[ScW00] Schudel, G. and B. Wood, "Adversary Work Factor As a Metric for Information Assurance," *Proceedings of the 2000 Workshop on New Security Paradigms,* pp. 23-30. Ballycotton, County Cork, Ireland, 2000.

[ShH02] Sheyner, O., J. Haines, S. Jha, R. Lippmann, and J. M. Wing, "Automated Generation and Analysis of Attack Graphs," *Proceedings of the 2002 IEEE Symposium on Security and Privacy,* pp. 273-284. Berkeley, California, 2002.

[ShL05] Shapiro, J., G. B. Lamont, and G. L. Peterson, "An Evolutionary Algorithm to Generate Hyper-Ellipsoid Detectors for Negative Selection," *Proceedings of the 2005 Genetic and Evolutionary Computation Conference,* Washington, D.C., 2005.

[Sna04] SnakeByte_(alias). "Win32 Bait Detection." http://madchat.org/vxdevl/papers/vxers/SnakeByte/w32bd.txt. Accessed 14 Nov 2005.

[Spa97] Spanksa_(alias). "ELVIRA virus." *29A Ezine*. http://www.madchat.org/vxdevl/vxmags/29a-2/29A%232.4_B. Accessed 28 Aug 2005.

[StS02] Steffan, J. and M. Schumacher, "Collaborative Attack Modeling," *Proceedings of the 2002 ACM Symposium on Applied Computing,* pp. 253-259. Madrid, Spain, 2002.

[SwB03] Swanson, M., N. Bartol, J. Sabato, J. Hash, and L. Graffo, *Security Metrics Guide for Information Technology Systems,* NIST Special Publication 800-55, National Institute of Standards and Technology, 2003.

[Szo05] Szor, P., *The Art of Computer Virus Research and Defense*. Upper Saddle River, NJ: Addison-Wesley, 2005.

[TiK04] Timmis, A. J., T. Knight, L. N. D. Castro, and E. Hart, "An Overview of Artificial Immune Systems," in *Computation in Cells and Tissues: Perspectives and Tools for Thought*, R. Paton, H. Bolouri, M. Holcombe, J. H. Parish, and R. Tateson, Eds. Berlin: Springer, pp. 51-86, 2004.

[TiL01] Tidwell, T., R. Larson, K. Fitch, and J. Hale, "Modeling Internet Attacks," *Proceedings of the 2001 IEEE Workshop on Information Assurance and Security,* pp. 54-59. United States Military Academy, West Point, NY, 2001.

[UnS05] *Lecture notes from School of Molecular and Microbial Sciences, University of Sydney, Australia*. www.biochem.usyd.edu.au/MBLG2001/BMedSci/lecture %2013_Transcription.ppt Accessed 30 Aug 2005.

[VaC94] Varadharajan, V. and C. Calvelli, "Extending the Schematic Protection Model - I: Authentication and Conditional Tickets," *Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy,* pp. 213-229. Oakland, CA, 1994.

[Vir04] "Virus Prevalence – 2004." http://www.virusbtn.com/resources/malware Directory/prevalence/index.xml?year=2004. Accessed 28 Aug 2005.

[Vir95] "Virus Prevalence – 1995." http://www.virusbtn.com/resources/malware Directory/prevalence/index.xml?year=1995. Accessed 28 Aug 2005.

[Wei91] Weiss, J. D., "A System Security Engineering Process," *Proceedings of the 14th National Computer Security Conference,* pp. 572-581. Washington DC, 1991.

[WiA01] Williams, P. D., K. Anchor, J. Bebo, G. Gunsch, and G. Lamont, "Warthog: Towards a Computer Immune System for Detecting “Low and Slow”

Information System Attacks," *Proceedings of the Recent Advances in Intrusion Detection,* Davis, CA, 2001.

[Wik05] "Self-modifying code." *Wikipedia*. http://en.wikipedia.org/wiki/Self-modifying _code. Accessed 1 Dec 2005.

[WoB01] Wood, B. and J. Bouchard, "Red Team Work Factor As a Security Measurement," *Proceedings of the 1st Workshop on Information-Security-System Rating and Ranking (WISSRR),* Williamsburg, Virginia, 2001.

[YaL97] Yao, X. and Y. Liu, "Fast Evolution Strategies," *Control & Cybernetics*, vol. 26, no. 3, pp. 467-496, 1997.

[YiZ04] Yip, S. and Z. Q. "Enhancing Software Protection with Poly-metamorphic Code." *New South Wales Society for Computers and the Law*. http://www.nswscl.org.au/journal/56/YipZhao.html. Accessed 1 Dec 2005.

[Yus05] Yuschuk, O. "OllyDbg." http://www.ollydbg.de. Accessed 1 Dec 2005.

[ZaL03] Zhang, Y., W. Lee, and Y. Huang, "Intrusion Detection Techniques for Mobile Wireless Networks," *Wireless Networks*, vol. 9, pp. 545-556, 2003.

Vita

Major Kenneth S. Edge graduated from Live Oak High School in Morgan Hill, California. He entered undergraduate studies at the United States Air Force Academy in Colorado Springs, Colorado where he graduated with a Bachelor of Science degree in Electrical Engineering and was commissioned a second lieutenant in June 1993.

His first assignment was at Sheppard AFB, Texas where he was a student in Euro-Nato Joint Jet Pilot Training. In May 1995, he was assigned to the 47th Airlift Flight, Wright-Patterson AFB, Ohio where he served as a C-21A Evaluator Pilot. While stationed at Wright-Patterson, he completed his Master of Science degree in Electrical Engineering at Wright State University. In August 1998, he was assigned to the 4th Airlift Squadron as a C-141B pilot and a Wing Flight Safety Officer. In September 2000, he was assigned to the 6th Airlift Squadron at McGuire AFB, New Jersey as a C-141B Instructor Pilot and Flight Commander. In May 2002, He was selected for Air Mobility Command's Phoenix Hawk Intern program and was stationed at Headquarters Air Mobility Command (HQ AMC), Scott AFB, Illinois as a Tanker Airlift Control Center Duty Officer and Chief of Aircrew Analysis in the Directorate of Operations. In August 2004, he entered the Graduate School of Engineering and Management, Air Force Institute of Technology.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From – To)*
13-09-2007	**Doctoral Dissertation**	Sep 2004 – Sep 2007

4. TITLE AND SUBTITLE

A Framework For Analyzing And Mitigating The Vulnerabilities Of Complex Systems Via Attack And Protection Trees

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Edge, Kenneth, S., Major, USAF

5d. PROJECT NUMBER

ENR # 07-152

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/DS/ENG/07-13

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/SNT
2241 Avionics Cir
WPAFB OH 45433

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Attack trees have been developed to describe processes by which malicious users attempt to exploit or break complex systems. Attack trees offer a method of decomposing, visualizing, and determining the cost or likelihood of attacks. Attack trees by themselves do not provide enough decision support to system defenders. This research develops the concept of using *protection trees* to offer a detailed risk analysis of a system. In addition to developing protection trees, this research improves the existing concept of attack trees and develops rule sets for the manipulation of metrics used in the security of complex systems.

This research specifically develops the framework for using an attack and protection tree methodology to analyze the security of complex systems. The structure of attack trees is extended and modified to create protection trees. To validate the effectiveness of the methodology, the Schematic Protection Model (SPM) is used. The SPM is extended and applied to verify that a system protected using the attack and protection tree methodology is safe. To demonstrate the general usefulness of this novel methodology, it is used to analyze the security of several varied domains including computer networks, online banking, homeland security, and mobile ad hoc networks.

15. SUBJECT TERMS

Protection Trees, Attack Trees, Risk Analysis, Computer Security

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
REPORT	ABSTRACT	c. THIS PAGE			Richard A. Raines (ENG)
U	U	U	UU	219	**19b. TELEPHONE NUMBER** *(Include area code)* (937) 255-6565, ext 4278; e-mail: Richard.Raines@us.af.mil